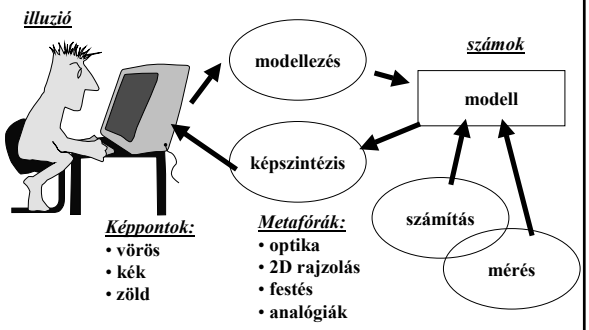
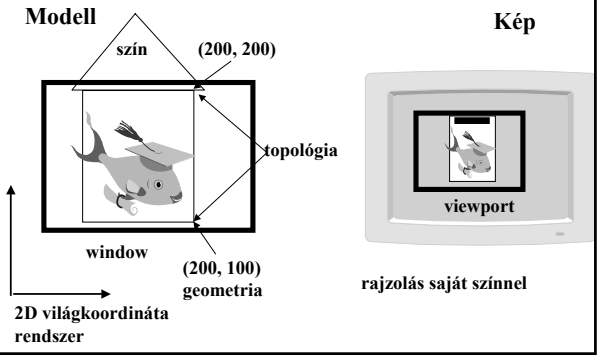
Számítógépes grafika

Szirmay-Kalos László
Irányítástechnika és Informatika Tanszék
email: szirmay@iit.bme.hu
Web: http://www.iit.bme.hu/~szirmay

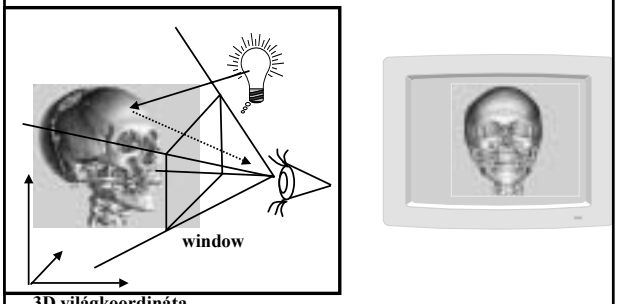
Számítógépes grafika feladata



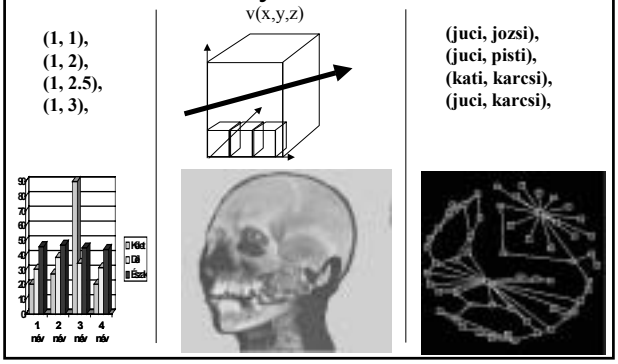
2D rajzolás, festés



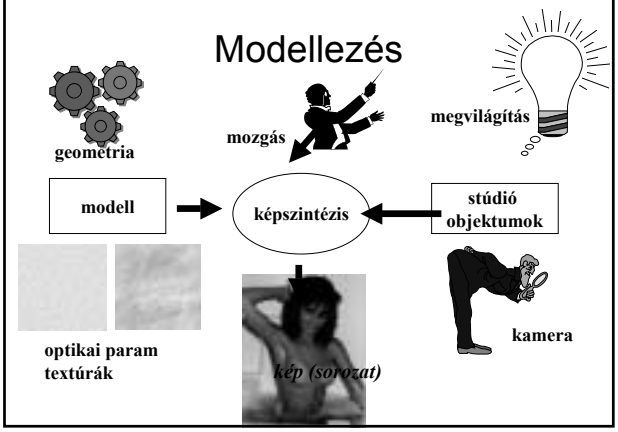
optika a 3D térben



Tudományos vizualizáció



Modellezés



Interaktivitás

koordináták inverz képszintézise
elemi modellváltoztatások

modellezés → modell

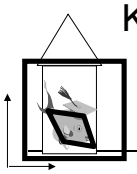
képszintézis → jelenlegi állapot

Undo:
- korábbi állapot (1 lép)
- beviteli parancsok

kiválaszthatóság

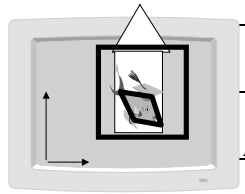
- grafikus beviteli eszközök: pont, pick
- elemi modellváltoztatás
- azonnali visszacsatolás
- Undo

Képszintézis feladatai



Vektorizáció

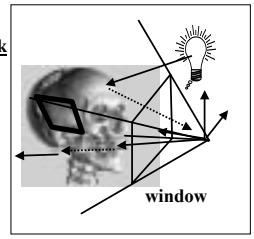
Transzformációk



Vágás

Takarás

Árnyalás



Étlap

- Számítógépes grafika elemei
- Grafikus hardware
- Grafikus software: esemény vezérelt programozás, MsWindows, X-Window, OpenGL
- Geometriai modellezés: görbék (spline), felületek, testek (B-Rep, CSG)
- Színek: színelmélet, színrendszerek, illesztés
- Transzformációk: homogén koordinátás alak, projektív geometria
- Modellek szerkezete
- 2D képszintézis: modellezési és ablak-nézet transzf, vágás, szakasz raszterizáció, területkitöltés, 2D grafikus rendszerek felépítése, pick
- Fraktálok: Hausdorff dim., Brown mozgás, Káosz a komplex síkon, Julia-Mandelbrot, IFS
- 3D képszintézis optikai alapmodellje: árnyalási egyenlet
- Lokális illuminációs algoritmusok: transzf, takarás (z-buffer), árnyalás (Gouraud, Phong)
- Sugárkövetés
- Globális illumináció: Monte-Carlo sugárkövetés, radiosity, ...
- Anti-aliasing: csipkék kiirtása
- Textúra leképzés
- Térfogat vizualizáció
- Animáció

Grafikus alaphardver

Szirmay-Kalos László

Jelfeldolgozási megközelítés

Analog | Digitál

Újramintavételezés
Újrakvantálás

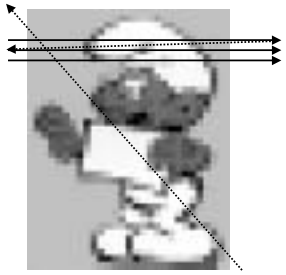
modellezés → modell → képszintézis → Digitális kép

megjelenítés

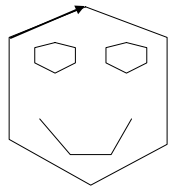
Analog | Digitál

Képinformáció építőelem
- vektor
- pixel

Raszteres - vektoros grafika

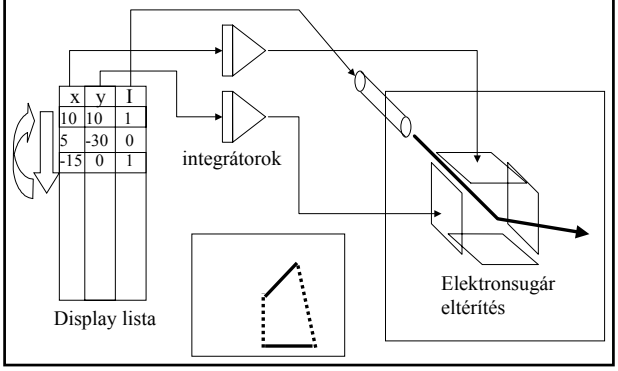


Aliasing, jaggies

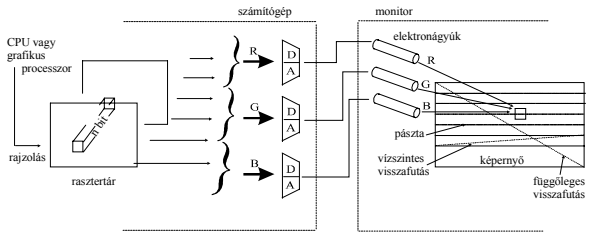


Rajzolási idő ~ vektorszám
1D elem: nincs kitöltött terület

Vektorgrafikus rendszerek



Rasztergrafikus rendszerek



Színkódolás

- Valós szín mód
  - rasztertár n bitjét három részre: *R, G, B*
  - választható színek = 2^n
  - egyszerre megjeleníthetô színek = 2^n
- Indexelt szín mód
  - raszter n bitje az RGB-t tartalmazó LUT címe
  - választható színek = 2^{3m}
  - egyszerre megjeleníthető színek = 2^n

Rasztertár sebességi problémái

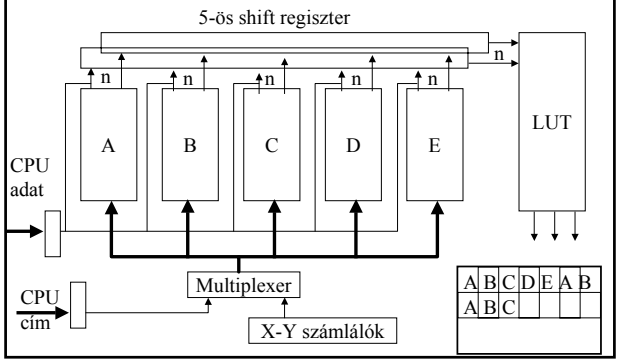
Rasztertár:
10^6 x 8…24 bit = 1..3Mbyte
sebesség: 80nsec

Rajzolás (animáció):
1 sec: 15...25-ször
pixelidő=1/15/10^6 = 66nsec

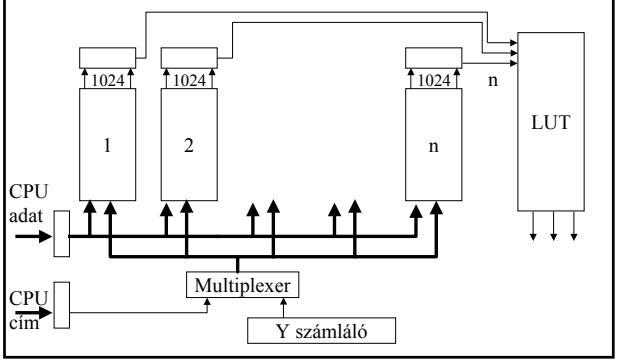
Megjelenítés:
1 sec: 50…100-szor
pixelidő=1/50/10^6 = 20nsec

Megoldás:
- Párhuzamos kiolvasás shiftregiszterbe
- Beíráshoz idő
  - kép, sorvisszafutás
  - frissítési hozzáférések között: 1024-es shiftreg.
- Párhuzamos beírás

5-blokkos memória



Video-RAM memória



Rendszertechnikai változatok

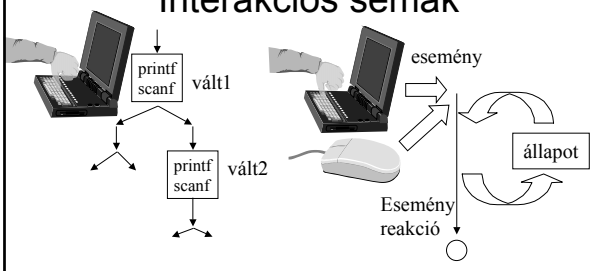
- PC modell
  - egy processzor, a rasztertár operatív memóriaként látszik
- Terminál modell
  - grafikus terminál, kommunikáció soros vonali parancsokkal
- Munkaállomás modell
  - speciális grafikus CPU, kommunikáció a grafikus CPU programjának letöltésével

Grafikus szoftver

Szirmay-Kalos László

Programok felépítése

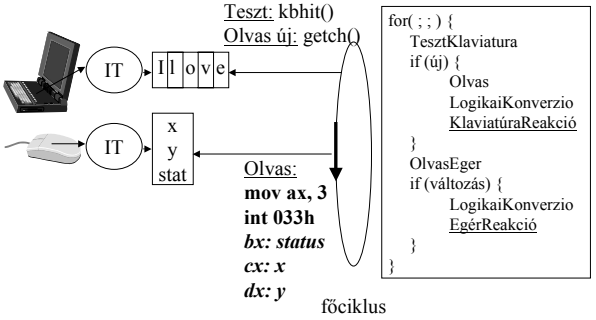
Interakciós sémák



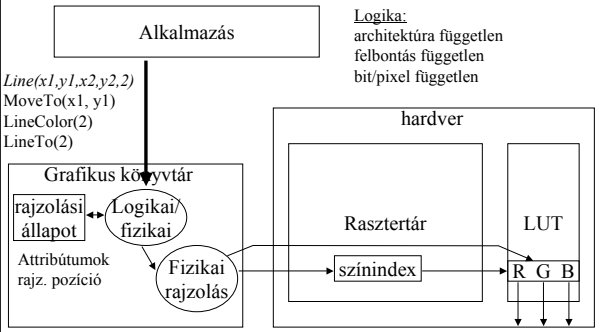
Programvezérelt:
fizikai esemény - változó összerendelés PC alapján

Eseményvezérelt:
fizikai esemény - változó összerendelés állapot alapján

Input kezelés (DOS)

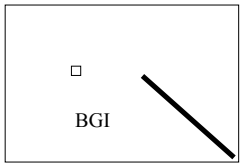


Output kezelés



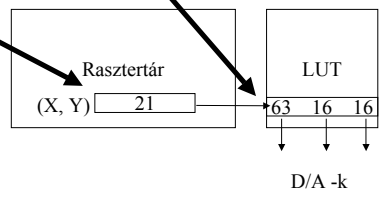
BGI grafika

```
#include <graphics.h>
...
int gdriver = DETECT, gmode;
initgraph(&gdriver, &gmode, ".");
if (graphresult() != grOK) exit( -1 );

setfillstyle( SOLID_FILL, BLACK );
bar( 0, 0, getmaxx( ), getmaxy( ) );

int color = RED, X = 100, Y = 100;
int c = getpixel( X, Y );
putpixel(X,Y, c ^ color);

moveto(200, 100);  setcolor( BROWN ); lineto( getmaxx( ), getmaxy( ) );
setcolor( WHITE ); outtextxy(100, 200, "BGI" );

closegraph();              // grafikus üzemmód kikapcsolása
```

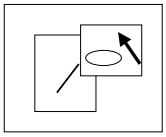


BGI színkezelés

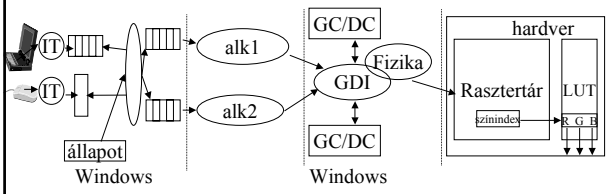
```
int pinkidx = 21;

setrgbpalette( pinkidx, 255, 64, 64 );
putpixel(X, Y, pinkidx );
```



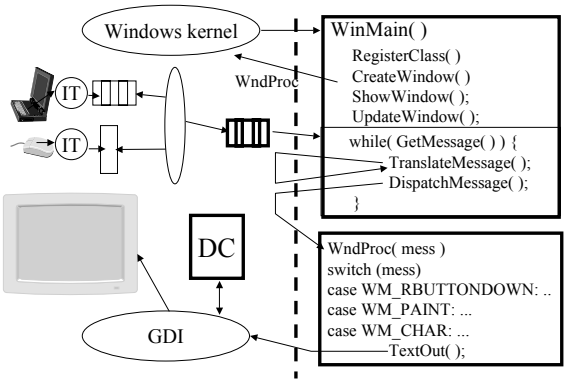
Ablakozott felhasználói felületek



- Asztalon könyvek metafórája
- Több alkalmazás
  - közös erőforráshasználat (input-output)
  - erőforrásokkal kapcsolatos események
- Input könyvtár
- Output könyvtár:
  - logikai (ablakrelatív) kezelés



MsWindows



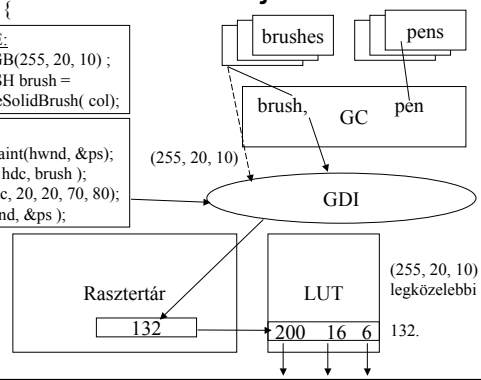
MsWindow állapot: DC

```
WndProc( mess )
    HDC hdc;

    switch (mess) {
    case WM_PAINT: ...
         hdc = BeginPaint(hwnd, &ps);
               attribútum állítás, rajzolás
          EndPaint( hwnd, &ps );
         break;

    case WM_RBUTTONDOWN: ..
    case WM_CHAR: ... összes többi
         hdc = GetDC( hwnd );
               attribútum állítás, rajzolás
         ReleaseDC( hwnd, hdc );
         break;
```

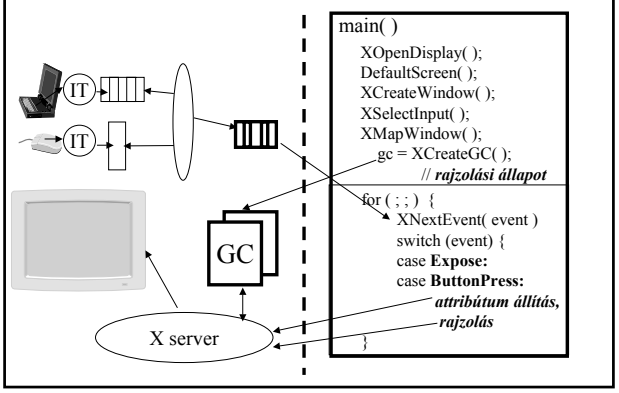
```
long col = RGB(255, 0, 0)
brush = CreateSolidBrush();
SelectObject( hdc, brush );
Rectangle( hdc, 20, 20, 70, 80);
DeleteObject( brush );
```

Ms-Windows rajzolás

```
switch (mess) {
WM_CREATE:
  long col = RGB(255, 20, 10) ;
  static HBRUSH brush =
          CreateSolidBrush( col);
WM_PAINT:
  hdc = BeginPaint(hwnd, &ps);
  SelectObject( hdc, brush );
  Rectangle( hdc, 20, 20, 70, 80);
  EndPaint( hwnd, &ps );
```



X-Window



X-Window: inicializálás

```
Display * dpy = XOpenDisplay (“”);                  // display megnyitás
int screen = DefaultScreen (dpy);                   // képernyő (lehet több is)
Visual * visual;
valuemask = CWBackPixel;

Window  win = XCreateWindow(dpy,
                    DefaultRootWindow (dpy), // szülô
                    x, y, width, height,     // hol, méretek
                    border_width,            // határ szélesség
                    depth,                   // bit pex pixel
                    InputOutput,             // ablak osztály
                    visual,                  // szín mód lekérdekés
                    valuemask,               // melyik attribútum kitöltött
                    &winattr);               // ablak attribútumok

XStoreName (dpy, win, ”I love IIT Tanszek” );       // ablak név
```

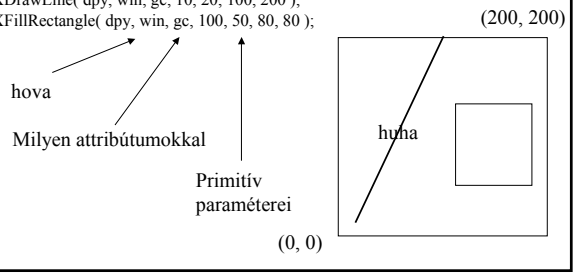
X-Window inicializálás + főciklus

```
                                                    // eseménysor szűrése
XSelectInput (dpy, win, StructureNotifyMask | ExposureMask | KeyPressMask );
XMapWindow (dpy, win);                              // ablak megjelenítés
gc = XCreateGC (dpy, win, 0L, (XGCValues *) 0);     // rajzolási állapot(ok)
XSetBackground (dpy, gc, BlackPixel (dpy, screen)); // index keresés
XSetForeground (dpy, gc, WhitePixel (dpy, screen));
```

```
for ( ; ; ) {
        XEvent event;
        XNextEvent (dpy, &event);                   // kiolvasás a sorból
        switch (event.type) {
        case ConfigureNotify:                       // az ablak mérete változott
                új méret: event.xconfigure.width, event.xconfigure.height
        case Expose: XClearWindow(dpy, win);        // az ablak újrarajzolást kér
        case KeyPress:    billentyű kód: event.xkey
        }
}
```

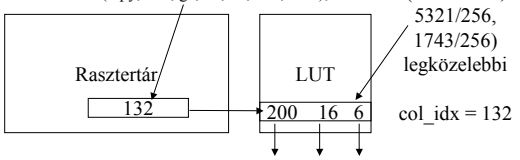
X-Window: rajzolás

```
char  text = ”huha”;
XDrawString( dpy, win, gc, 50, 100, text, strlen(text) );
XSetForeground (dpy, gc, WhitePixel (dpy, screen));
XDrawLine( dpy, win, gc, 10, 20, 100, 200 );
XFillRectangle( dpy, win, gc, 100, 50, 80, 80 );
```



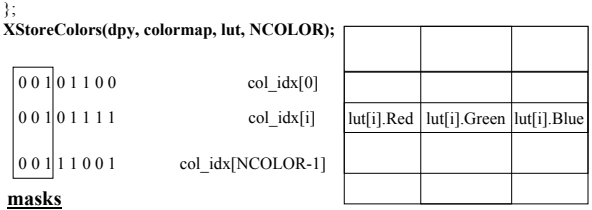
Színek a meglévőből

```
colormap = XDefaultColormap(dpy, screen );
XColor col;
col.red = 65535, col.green = 5321, col.blue = 1743;
col.flags = DoRed | DoGreen | DoBlue;
if ( !XAllocColor( dpy, colormap, &col ) ) Nem tud adni …
long col_idx = col.pixel;
XSetForeground (dpy, gc, col_idx);
XDrawLine( dpy, win, gc, 10, 20, 100, 200 );
```



Új színek allokálása

```
long col_idx[NCOLOR], masks[NPLANE];
colormap = XDefaultColormap(dpy, screen );
XAllocColorCells (dpy, colormap, 0, &mask, 0, col_idx, NCOLOR);
for(int i=0; i <NCOLOR; i++) {
        lut[i].pixel = col_idx[i];
        lut[i].flags = DoRed | DoGreen | DoBlue;
        lut[i].red = ...; lut[i].green = ... ; lut[i].blue;
};
XStoreColors(dpy, colormap, lut, NCOLOR);
```



OpenGL: inicializálás

```
glutInitWindowSize(wwidth, wheight );
glutInitWindowPosition(wposx, wposy );
glutInit(&argc, argv);
glutInitDisplayMode( GLUT_RGBA );

glutCreateWindow("Sample Window");
        // callback függvények
glutKeyboardFunc( Keyboard );
glutDisplayFunc( ReDraw );
        // transzformáció
glViewport(0, 0, 200, 200);
glLoadIdentity();
gluOrtho2D(0.0, 100.0, 0.0, 100.0);
        // fô hurok
glutMainLoop();
```

KeyPress

Expose

(100,100) window (0,0) → (200,200) viewport (0,0)

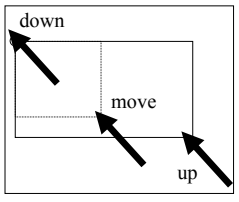
OpenGL: eseménykezelés és rajzolás

```
void ReDraw( ) {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
}

void Keyboard(unsigned char key, int x, int y) {
    if (key == 'd') {
        glColor3d( 1.0, 0.0, 0.0 );
        glBegin(GL_TRIANGLES);;
        glVertex2d(10.0, 10.0);
        glVertex2d(20.0, 100.0);
        glVertex2d(80.0, 30.0);
        glEnd( );
        glFlush( );
    }
}
```

window (100,100) (0,0)

viewport (200,200) (0,0)

Eseményvezérelt program tervezése, példa: gumi négyszögek



DrawRubber
DelRubber
XOR

Felhasználó — Program

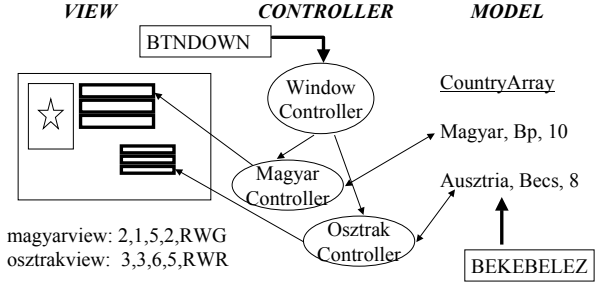
	IDLE	
MouseLeftBtnDown	→ *START*	
MoveMouse	→ *RUBBER*	DrawRubber
MoveMouse *	→ *RUBBER*	DelRubber DrawRubber
MouseLeftBtnUp	→ *IDLE*	DelRubber DrawRect

Állapotgép

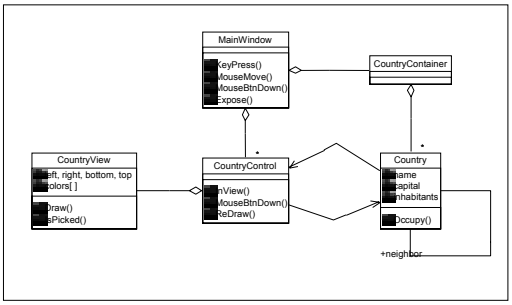
esemény / állapot	MouseLeft BtnDown	Mouse Move	MouseLeft BtnUp
IDLE	xstart=, ystart= START		
START		xend=, yend= DrawRubber RUBBER	IDLE
RUBBER		DelRubber xend=, yend= DrawRubber RUBBER	DelRubber xend=, yend= Draw IDLE

```
static int xe, ye, xs, ys;
static enum {IDLE, START, RUBBER} state = IDLE;

WndProc( HWND hwnd, WORD msg, WORD wpar, LONG lpar) {
    switch( wmsg) {
    case WM_LBUTTONDOWN: xs = LOWORD(lpar); ys = HIWORD(lpar); break;
    case WM_MOUSEMOVE:
        switch (state) {
        case START:     xe = LOWORD(lpar); ye = HIWORD(lpar);
                        DrawRubber( );   state = RUBBER; break;
        case RUBBER:    DelRubber( );
                        xe = LOWORD(lpar); ye = HIWORD(lpar);  DrawRubber( );
        }
    case WM_LBUTTONUP:
        switch (state) {
        case START:     state = IDLE; break
        case RUBBER:    DelRubber( ); xe = LOWORD(lpar); ye = HIWORD(lpar);
                        Draw( );  state = IDLE;
        }
    }
}
```

UID - alkalmazás szétválasztása: MVC paradigma



MVC: UML osztálydiagram



Geometriai modellezés

Szirmay-Kalos László

Számítógépes grafika elemei

- modellezés
- virtuális világ
- képszintézis

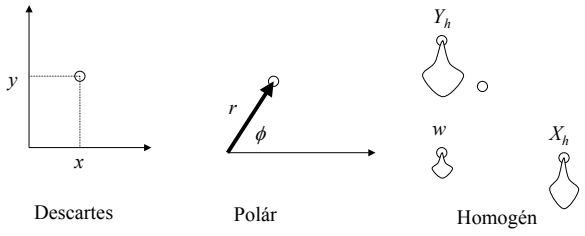
Modellezés feladatai

- Geometria megadása
  - pont, görbe, terület, felület, test, fraktálok
- Transzformációk
  - lokális modellezési és világkoordinátarendszer
- Színek, felületi optikai tulajdonságok

Pontok megadása

- Mindent számmal!
  - Koordináta rendszer
  - Koordináták megadása
- Koordináta rendszerek
  - Descartes
  - Polár
  - Homogén

Koordináta rendszerek

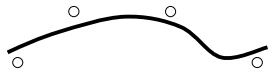


Görbék

- Görbe pontok halmaza:
  - koordinátáik kielégítenek egy egyenletet
  - implicit: $f(x, y) = 0$
    - Kör: $(x - x0)^2 + (y - y0)^2 - r^2 = 0$
  - explicit: $x = x(t), y = y(t), t \in [0,1]$
    - Kör: $x = x0 + r \cos 2\pi t$, $y = y0 + r \sin 2\pi t \; t \in [0,1]$
- Klasszikus görbék
  - van egyenlet
  - definíció = paraméterek megadása

Szabadformájú görbék

- Definíció kontrolpontokkal



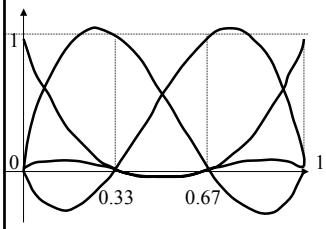
- Polinom: $x(t) = \Sigma\, a_i\, t^i$, $y(t) = \Sigma\, b_i\, t^i$
- A polinomegyütthatók származtatása:
  - Interpoláció
  - Approximáció

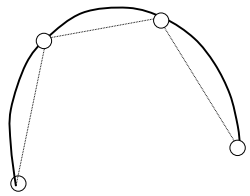
Lagrange interpoláció

- Kontrolpontok: $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \ldots, \mathbf{r}_n$
- Keressük azt a $\mathbf{r}(t) = \Sigma\, [a_i, b_i]\, t^i$-t amelyre $\mathbf{r}(t_1) = \mathbf{r}_1, \mathbf{r}(t_2) = \mathbf{r}_2, \ldots, \mathbf{r}(t_n) = \mathbf{r}_n$,
- Megoldás:
  - $\mathbf{r}(t) = \Sigma\, L_i(t)\, \mathbf{r}_i$

$$L_i(t) = \frac{\Pi_{j \neq i}\,(t-t_j)}{\Pi_{j \neq i}\,(t_i-t_j)}$$

Lagrange interpoláció bázisfüggvényei





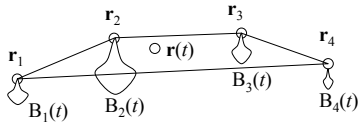
$$L_i(t) = \frac{\Pi_{j \neq i}\,(t-t_j)}{\Pi_{j \neq i}\,(t_i-t_j)}$$

$$\mathbf{r}(t) = \Sigma\, L_i(t)\, \mathbf{r}_i$$

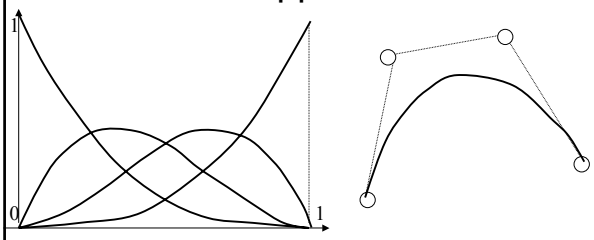
Görbeszerkesztés Lagrange interpolációval

Bezier approximáció

- Keresett görbe: $\mathbf{r}(t) = \Sigma\, B_i(t)\, \mathbf{r}_i$
  - $B_i(t)$: ne okozzon nem indokolt hullámokat
  - Konvex burok tulajdonság
  - $B_i(t) \geq 0, \quad \Sigma\, B_i(t) = 1$



Bezier approximáció

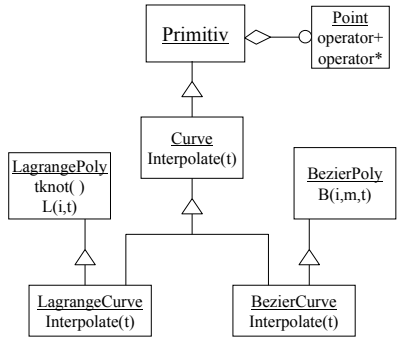


$$B_i(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

$$\mathbf{r}(t) = \Sigma\, B_i(t)\, \mathbf{r}_i$$

Bernstein polinomok

Görbék C++ -ban



Primitív és Curve

```
class Primitive2D {
   Array<Point2D> points;
   Color          color;
public:
   Primitive2D( Color& c, int n = 0 ) : color(c), points(n) { }
   Point2D& Point( int i ) { return points[i]; }
   int     PointNum( ) { return points.Size(); }
};

class Curve2D : public Primitive2D {
public:
   Curve2D( Color& c ) : Primitive2D( c ) { }
   virtual Point2D Interpolate( double tt ) = 0;
};
```

Lagrange polinom

```
class LagrangePolinom {
   Array<double> knot_pars;
public:
   int Degree( ) { return knot_pars.Size(); }
   double& t( int i ) { return knot_pars[i]; }
   double L( int i, double tt ) {
      double Li = 1.0;
      for(int j = 0; j < Degree(); j++)  if (i != j) Li *= (tt - t(j)) / (t(i) - t(j));
      return Li;
   }
   void DistributeKnotPars( int n ) {  for (int i = 0; i <= n; i++) t(i) = (double)i / n; }
};
```

LagrangeCurve

```
class LagrangeCurve2D : public Curve2D, public LagrangePolinom {
public:
   LagrangeCurve2D( Color c ) : Curve2D( c ), LagrangePolinom( ) { }
   Point2D Interpolate( double tt ) {
      Point2D rr(0, 0);
      for(int i = 0; i < Degree(); i++) rr += Point(i) * L(i, tt);
      return rr;
   }
};
```

Bezier polinom

```
class BezierPolinom {
public:
   double B( int i, double tt, int m ) {
      double Bi = 1.0;
      for(int j = 1; j <= i; j++)  Bi *= tt * (m-j)/j;
      for(       ; j <  m; j++)  Bi *= (1-tt);
      return Bi;
   }
};
```

BezierCurve

```
class BezierCurve2D : public Curve2D, public BezierPolinom {
public:
   BezierCurve2D( Color c ) : Curve2D( c ), BezierPolinom( ) { }
   Point2D Interpolate( double tt ) {
      double Bi = 1.0;
      Point2D rr(0, 0);
      for(int i = 0; i < PointNum(); i++)
         rr += Point(i) * B(i, tt, PointNum());
      return rr;
   }
};
```

Bonyolult görbék

- Nagyon magas fokszámú polinom
- Összetett görbék:
  - Több alacsony fokszámú + folytonos illesztés
  - folytonossági kategóriák

C^0: $\mathbf{r}_1(t_{veg}) = \mathbf{r}_2(t_{kezd})$

C^1: $\mathbf{r}_1'(t_{veg}) = \mathbf{r}_2'(t_{kezd})$

C^2: $\mathbf{r}_1''(t_{veg}) = \mathbf{r}_2''(t_{kezd})$

$G^0 = C^0$ $G^1 \supset C^1$

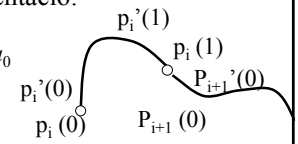
Spline

- Spline: C^2 folytonos összetett görbe
  - Harmadfokú spline
  - B-spline

Harmadfokú spline

- $p(t) = a_3 t^3 + a_2 t^2 + a_1 t^1 + a_0$
- Új szemléletes reprezentáció:

$p(0) = a_0$
$p(1) = a_3 + a_2 + a_1 + a_0$
$p'(0) = a_1$
$p'(1) = 3a_3 + 2a_2 + a_1$



- C^1 folytonosság: 2 paraméter közös
- C^2 folytonosság: ~~$p_i''(1) = p_{i+1}''(0)$~~

B-spline

- Válasszunk olyan reprezentációt, amely C^2 folytonos, ha 3-t közösen birtokolnak
- Reprezentáció: kontrolpontok

$$\mathbf{r}^{i}(t) = B_0(t)\mathbf{r}_0 + B_1(t)\mathbf{r}_1 + B_2(t)\mathbf{r}_2 + B_3(t)\mathbf{r}_3$$

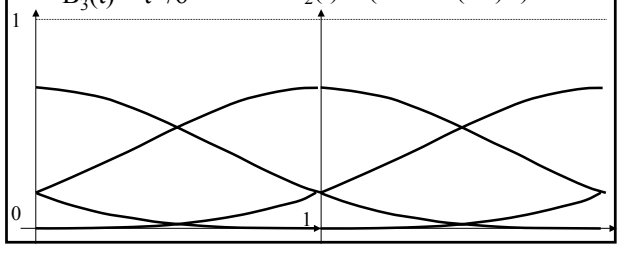
$$\mathbf{r}^{i+1}(t) = B_0(t)\mathbf{r}_1 + B_1(t)\mathbf{r}_2 + B_2(t)\mathbf{r}_3 + B_3(t)\mathbf{r}_4$$

B-spline bázisfüggvények

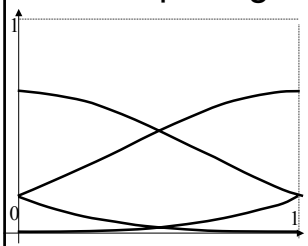
- Cirkuszi elefántok + Járulékos szempont: $\Sigma B_i(t) = 1$

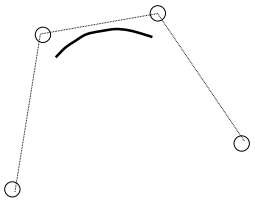
$B_0(t) = (1-t)^3/6$ $B_1(t) = (1+3(1-t)+3t(1-t)^2)/6$

$B_3(t) = t^3/6$ $B_2(t) = (1+3t+3(1-t)t^2)/6$



B-spline görbeszegmens

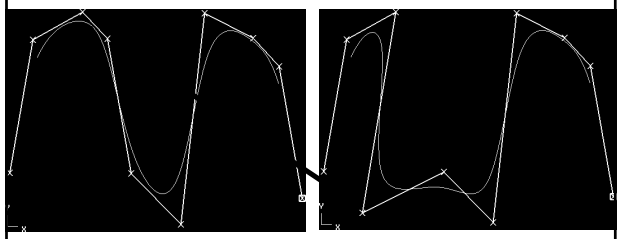




$B_0(t) = (1-t)^3 /6$ $B_1(t) = (1+3(1-t)+3t(1-t)^2) /6$

$B_3(t) = t^3 /6$ $B_2(t) = (1+3t+3(1-t)t^2) /6$

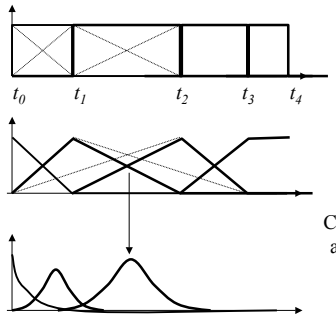
A B-spline lokálisan vezérelhetô

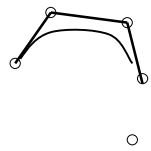


NUBS: Non-Uniform B-spline

- B-spline
  - minden szegmens 1 hosszú paramétertartomány
  - Akkor megy át a kontrol ponton, ha három egymás követő kontrolpont egymásra illeszkedik
- NUBS
  - az *i*. szegmens t_i -től t_{i+1} -ig.
  - Egy kontrolpont többször is számíthat:
    - A legalább 3-szoros pontokon a görbe átmegy

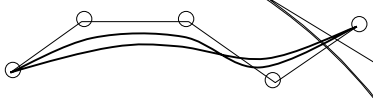
NUBS: Non-Uniform B-spline





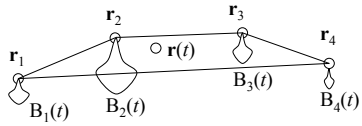
Cox-deBoor algoritmus

NUBS program



```
B(i, k, t) {                    // 0/0 = 0.
   if (k == 0)  { if (t[i] <= t < t[i+1]) return 1  else  return 0; }
   else return  ( (   t  -  t[i] ) * B(i, k-1, t )    /    (   t[i+k] - t[i]    ) +
                  (t[i+k+1]-t) * B(i+1, k-1, t) / (t[i+k+1] - t[i+1]) );
}
NUBS(k, t) {
   r = 0;  for(i = 0; i < N; i++) r += r[i] * B(i, k, t);  return r;
}
```

Idáig: Nem racionális B-spline



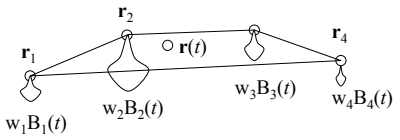
Idáig a súlyfüggvények: $\Sigma\, B_i(t)$

Polinom!

Súlypont:

$$\mathbf{r}(t) = \frac{\Sigma\,(B_i(t)\,\mathbf{r}_i)}{\Sigma\,B_i(t)} = \Sigma\,B_i(t)\,\mathbf{r}_i$$

NURBS: Non-uniform Rational B-spline



Polinom tört! racionális

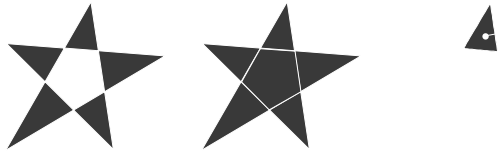
$$\mathbf{r}(t) = \frac{\Sigma\,(w_i B_i(t)\,\mathbf{r}_i)}{\Sigma\,w_i B_i(t)} = \Sigma \boxed{\frac{w_i B_i(t)}{\Sigma\,w_i B_i(t)}}\,\mathbf{r}_i$$

$B_i^*(t)$

Területek

Határ + belső tartományok azonosítása

Belső tartományok:



Felületek

- Felület 3D pontok halmaza:
  - koordinátáik kielégítenek egy egyenletet
  - implicit: $f(x, y, z) = 0$
    - gömb: $(x - x0)^2 + (y - y0)^2 + (z - z0)^2 - r^2 = 0$
  - explicit: $x = x(u,v)$, $y = y(u,v)$, $u,v \in [0,1]$
    - gömb: $x = x0 + r \cos 2\pi u \sin \pi v$
      $y = y0 + r \sin 2\pi u \sin \pi v$
      $z = z0 + r \cos \pi v \qquad u,v \in [0,1]$
- Klasszikus felületek
  - definíció = paraméterek megadása

Kvadratikus felületek

- $\underline{x}^T \mathbf{A}\, \underline{x} = 0 \qquad \underline{x}^T = [x, y, z, 1]$
- A koordináták legfeljebb másodfokon
- gömb, ellipszoid, sík, paraboloid, hiperboloid, hengerfelület,...

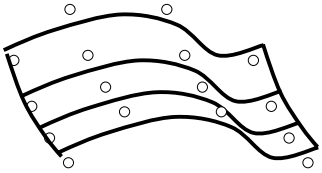
Ellipszoid $\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}-1=0$

Végtelen kúp $\frac{x^2}{a^2}+\frac{y^2}{b^2}-z^2=0$

Végtelen henger $\frac{x^2}{a^2}+\frac{y^2}{b^2}-1=0$

Szabadformájú felületek

- Definíció kontrolpontokkal, kontrolgörbékkel



Súlyfüggények: Interpoláció, Approximáció

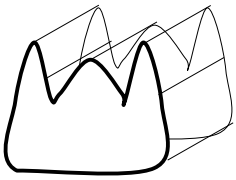
$$\mathbf{r}(u,v) = \Sigma\,\Sigma\, B_{i,j}(u,v)\,\mathbf{r}_{i,j}$$
$$= \Sigma\,\Sigma\, B_i(u)\,B_j(v)\,\mathbf{r}_{i,j}$$

szorzatfelületek

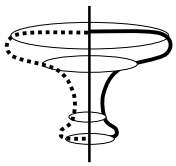
Testek

- Érvényes testek: reguláris halmaz
  - nem lehetnek alacsony dimenziós elfajulásai
  - minden határpont mellett van belső pont
- Garantáltan érvényes testet építő módszerek
  - 2.5 dimenziós eljárások
  - speciális felületi modellezés: B-rep
  - Konstruktív tömörtest geometria

2.5 dimenziós módszerek



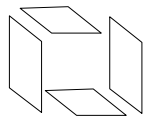
Kihúzás: extrude

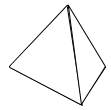


Rotate: forgatás

Felületmodellezők

- Test = határfelületek gyűjteménye





- Topológiai ellenőrzés (Euler tétel):

csúcs + lap = él + 2

B-rep

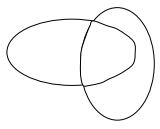
- Felületi modellező
- Elemi műveletek nem sérthetik a topológiát
  - Euler operátorok
- A teljes topológiai információt tároljuk
  - szárnyas él adatstruktúra

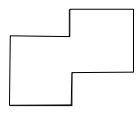
B-rep: Euler operátorok

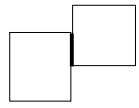
Tetraéder Euler operátorokkal

CSG: konstruktív tömörtest geometria

- 3D ponthalmazok uniója, metszete … is ponthalmaz







Kizárás:
reguális halmazműveletek

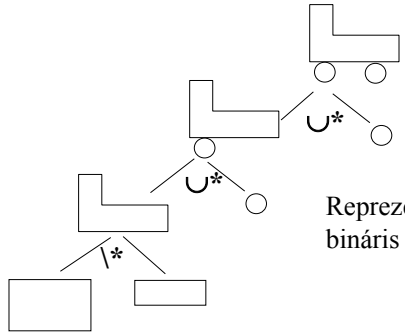
CSG alapműveletek

Reguláris unió

Reguláris különbség

Reguláris interszekció

CSG modellezés



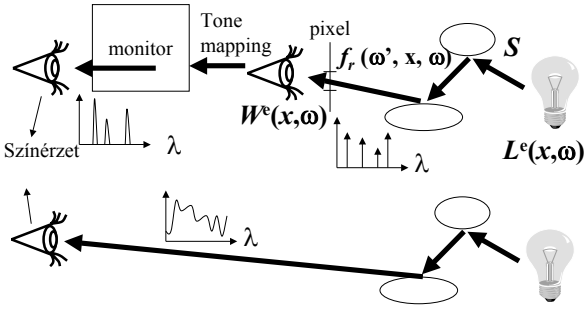
Reprezentáció: bináris fa

Normál és ciklikus CSG

Színek

Szirmay-Kalos László

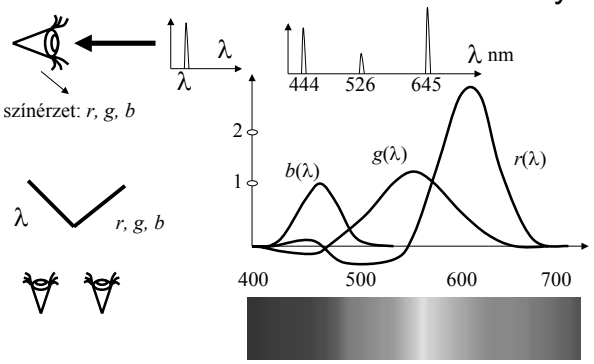
Képszintézis: valós világ látásának illuziója



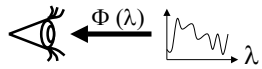
Színelméleti alapok

1. Fény - Színérzet
   - elektromágneses hullám,
   - érzéklés 3 érzékelővel: tristimulus értékek
   - Hullámhosszak: 700 nm, 561nm, 436 nm

Színérzékelés: monokromatikus fény

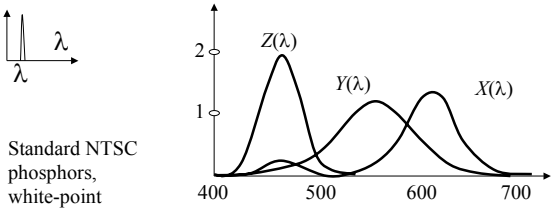


Színérzékelés: nem monokromatikus fény



$$r = \int \Phi(\lambda)\, r(\lambda)\, d\lambda$$
$$g = \int \Phi(\lambda)\, g(\lambda)\, d\lambda$$
$$b = \int \Phi(\lambda)\, b(\lambda)\, d\lambda$$

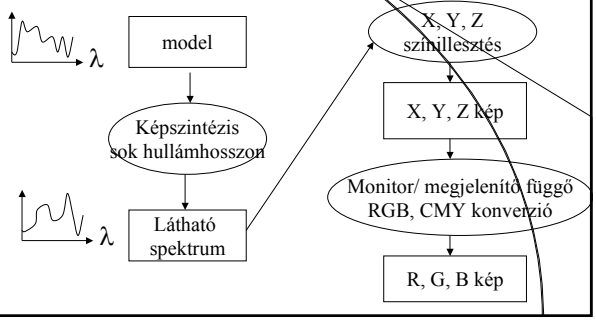
Pozitív súlyú bázis: XYZ



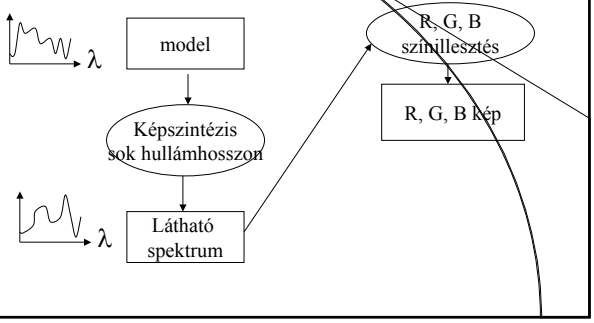
Standard NTSC phosphors, white-point

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.967 & -0.548 & -0.197 \\ -0.955 & 1.938 & -0.027 \\ 0.064 & -0.130 & 0.982 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

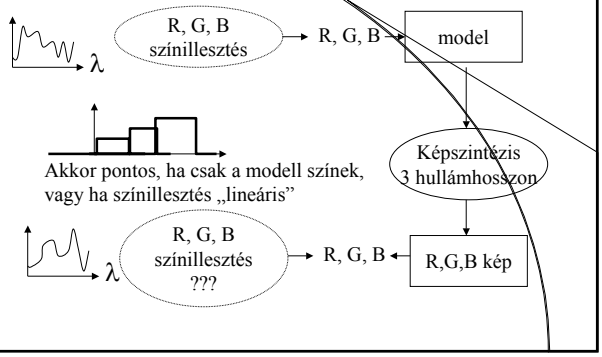
Színkezelés a számítógépes grafikában



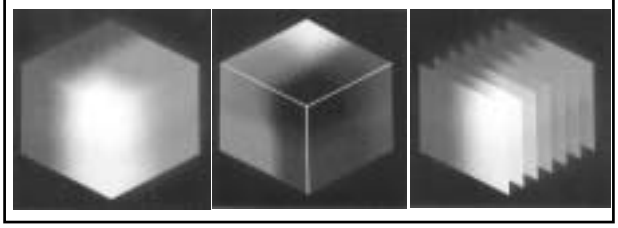
Megjelenítőfüggetlen színkezelés



Nem spektrális színkezelés



Színek definiálása



CMY színrendszerről RGB-re

```
void Color :: CMYToRGB( double C, double M, double Y ) {
        R = 1.0 - C;
        G = 1.0 - M;
        B = 1.0 - Y;
}
```

HLS színrendszer

L=0.1 L=0.5 L=0.8

HLS-ről RGB-re

```
double Color :: HexaCone( double s1, double s2, double hue ) {
   while (hue > 360)   hue -= 360;
   while (hue < 0)     hue += 360;
   if (hue < 60)   return (s1 + (s2 - s1) * hue/60);
   if (hue < 180)  return (s2);
   if (hue < 240)  return (s1 + (s2 - s1) * (240-hue)/60);
   return (s1);
}

void Color :: HLSToRGB( double H, double L, double S ) {
   double s2 = (L <= 0.5) ? L * (1 + S) : L * (1 - S) + S,  s1 = 2 * L - s2;
   if (S == 0) { R = G = B = L; }
   else {
      R = HexaCone(s1, s2, H - 120);
      G = HexaCone(s1, s2, H);
      B = HexaCone(s1, s2, H + 120);
   }
}
```

Geometriai Transzformációk

Szirmay-Kalos László

Geometriai transzformációk

- Geometriai reprezentáció váltás
- Függvény az (x,y) koordinátákon
- Affin transzformációk: párhuzamos tartás
- lineáris transzformációk: lineáris függény

Elemi transzformációk

- Eltolás: $\mathbf{r} = \mathbf{r} + \mathbf{p}$
- Skálázás: $x' = S_x\, x; \quad y' = S_y\, y;$

Fix pont: origó

$$\mathbf{r'} = \mathbf{r} \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

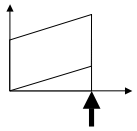
- Forgatás:

Fix pont: origó

$$\mathbf{r'} = \mathbf{r} \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix}$$

Elemi transzformációk

- Nyírás: $x' = x; \quad y' = y + a\, x;$



$$\mathbf{r'} = \mathbf{r} \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}$$

- Tükrözés:

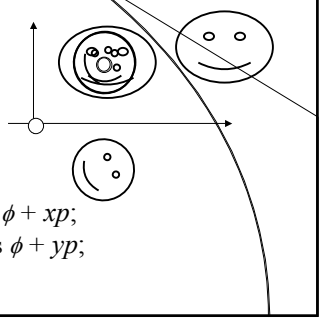
$$\mathbf{r'} = \mathbf{r} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Transzformáció fix pontja: pivot point: (xp, yp)

Skálázás:

$x' = S_x\,(x - xp) + xp;$

$y' = S_y\,(y - yp) + yp;$



Forgatás:

$x' = (x - xp) * \cos\phi - (y - yp) * \sin\phi + xp;$

$y' = (x - xp) * \sin\phi + (y - yp) * \cos\phi + yp;$

Összetett transzformáció

- Lineáris transzformáció: $\mathbf{r'} = \mathbf{r}\,\mathbf{A} + \mathbf{p}$
  - **A**: forgatás, skálázás, tükrözés, nyírás, stb.
  - **p**: eltolás
- Amíg nincs forgatás: konkatenáció
  - $\mathbf{r'} = (\ldots(\mathbf{r}\,\mathbf{A_1})\,\mathbf{A_2})\ldots\,\mathbf{A_n}) = \mathbf{r}\,(\mathbf{A_1}\mathbf{A_2}\ldots\,\mathbf{A_n})$

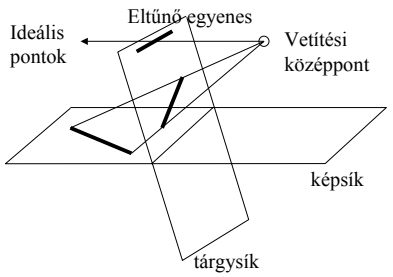
Homogén koordinátás transzformációk

- Eltolás nem fér bele a 2x2-es mátrixba
  - Dolgozzunk 3x3-as mátrixokkal

$$[\mathbf{r'}, 1] = [\mathbf{r}, 1] \begin{bmatrix} \overbrace{\begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{matrix}}^{\mathbf{A}} & \begin{matrix} 0 \\ 0 \end{matrix} \\ \underbrace{\begin{matrix} p_1 & p_2 \end{matrix}}_{\mathbf{p}} & 1 \end{bmatrix} = [\mathbf{r}\,\mathbf{A} + \mathbf{p}, 1]$$

$$[\mathbf{r'}, 1] = (\ldots([\mathbf{r}, 1]\,\mathbf{T_1})\,\mathbf{T_2})\ldots\,\mathbf{T_n}) = [\mathbf{r}, 1]\,(\mathbf{T_1}\mathbf{T_2}\ldots\,\mathbf{T_n})$$

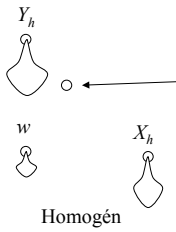
Centrális projekció



Projektív geometria

- Eukleideszi geometria
  - centrális projekcióra lyukas (idális pontok)
  - algebrai alap: Descartes koordináta rendszer
- Projektív geometria =
  = Eukleideszi geometria + ideális pontok
  - algebrai alap: homogén koordináták

Homogén koordináták

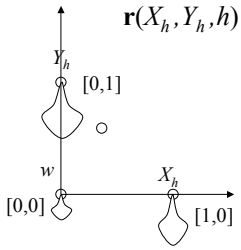


Összsúly: $h = X_h + Y_h + w$

Pont homogén koordinátái:

$$(X_h, Y_h, h)$$

Homogén-Descartes kapcsolat affin pontokra



$$\mathbf{r}(X_h, Y_h, h) = \frac{X_h[1,0] + Y_h[0,1] + w[0,0]}{h}$$

$$\mathbf{r}(X_h, Y_h, h) = \left[\frac{X_h}{h}, \frac{Y_h}{h}\right]$$

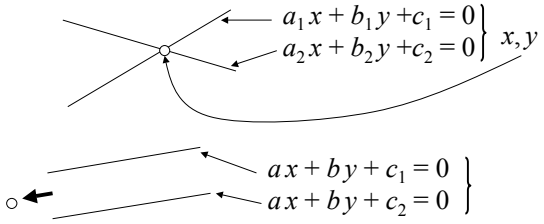
$$x = \frac{X_h}{h} \qquad y = \frac{Y_h}{h}$$

Következmények

- Minden affin ponthoz van: (X_h, Y_h, h)
  - $[x, y] \Rightarrow (x, y, 1)$
- Ha $h \neq 0$, akkor (X_h, Y_h, h) affin pont

$$\left[\frac{X_h}{h}, \frac{Y_h}{h}\right]$$

Párhuzamos egyenesek metszéspontja Descartes koordinátákkal



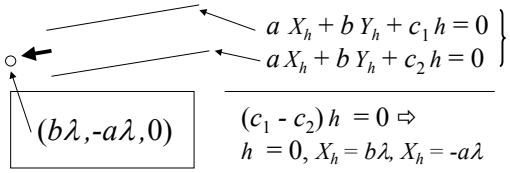
$c_1 - c_2 = 0 \Rightarrow$ nincs megoldás

Párhuzamos egyenesek metszéspontja homogén koordinátákkal

Descartes: $a x + b y + c = 0$

$a X_h/h + b Y_h/h + c = 0$

Homogén: $a X_h + b Y_h + c h = 0$



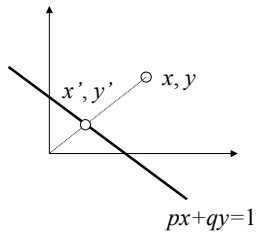
$(c_1 - c_2) h = 0 \Rightarrow$
$h = 0, X_h = b\lambda, X_h = -a\lambda$

Homogén lineáris transzformációk

- Euklideszi tér lineáris transzformáció:
  $[x', y'] = [x, y]\ \mathbf{A} + \mathbf{p}$
- Projektív tér lineáris transzformációi:
  $(X_h', Y_h', h') = (X_h, Y_h, h)\ \mathbf{T} + \mathbf{p}$
- Homogén lineáris transzformációk bővebbek:

$$\mathbf{T} = \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ p_1 & p_2 & 1 \end{bmatrix}$$

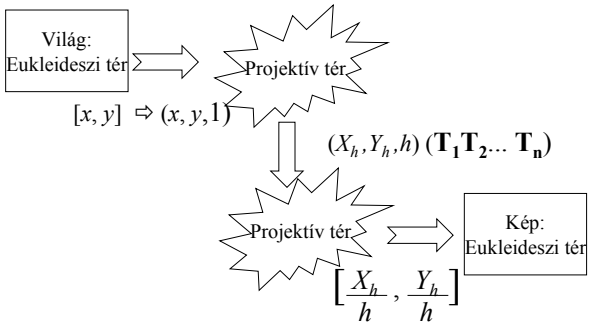
Példa: Euklideszi geometriában nem lineáris transzformáció



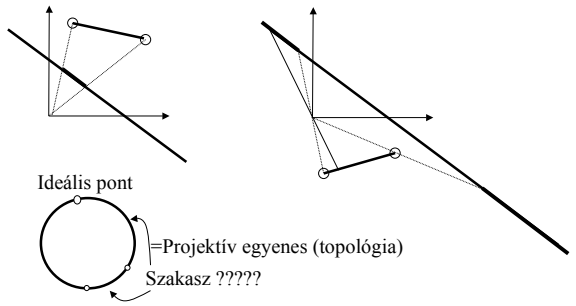
$$[x, y, 1] \begin{bmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 0 \end{bmatrix} \quad [x, \quad y, \ px+qy]$$

$$\left[\frac{x}{px+qy} \quad \frac{y}{px+qy} \right]$$

Projektív geometria a számítógépes grafikában



Veszélyek: átfordulási probléma

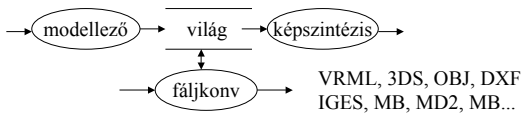


Virtuális világ tárolása

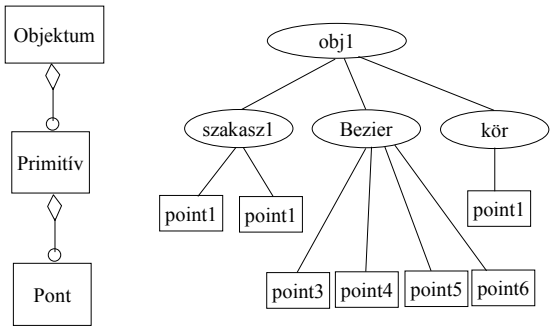
Szirmay-Kalos László

Belső világ tárolása

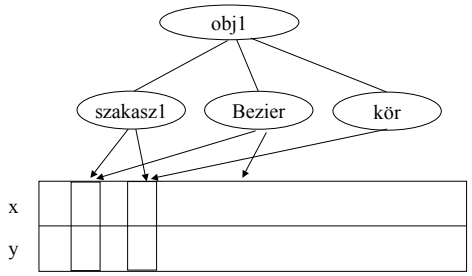
- Geometria: pontok koordinátái
- Topológia: élek-pontok; lapok-pontok;...
- hierarchia: objektum-lapok-élek-pontok
- transzformáció: lokális és világkoordináta rendszerek



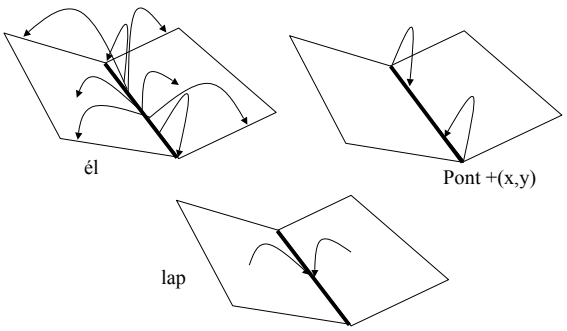
Egyszerű hierarchikus modell



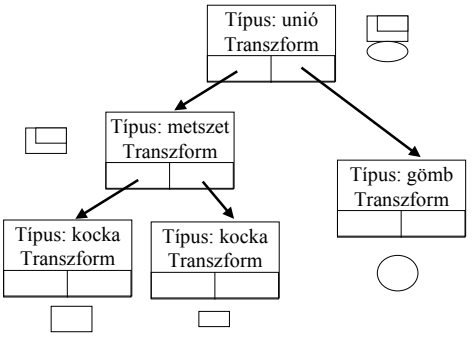
Geometria kiemelése



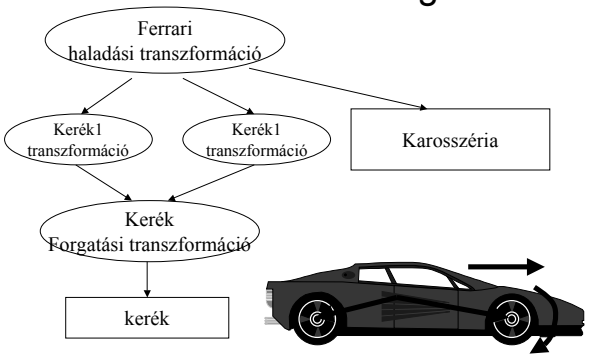
Teljes topológia: szárnyas él



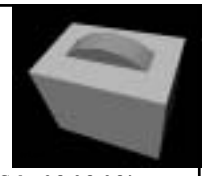
CSG fa



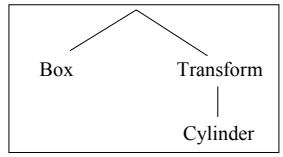
Hierarchikus színtér gráfok



VRML

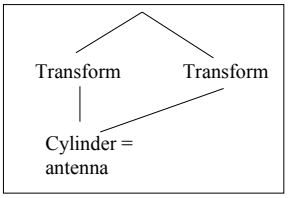


```
#VRML V2.0 utf8

Shape {
  geometry Box { size 1.5  1.2  1 }
  appearance Appearance { material Material { diffuseColor 0.8  0.8  0.8 }
  }
}

Transform {
  translation 0  0.1  0
  rotation 1 0 0   1.571
  children [
     Shape {
          geometry Cylinder { radius 0.7  height 0.3 }
          appearance Appearance { material Material { diffuseColor 0.7 0.7 0.1 }}
     }
  ]
}
```



VRML folytatás

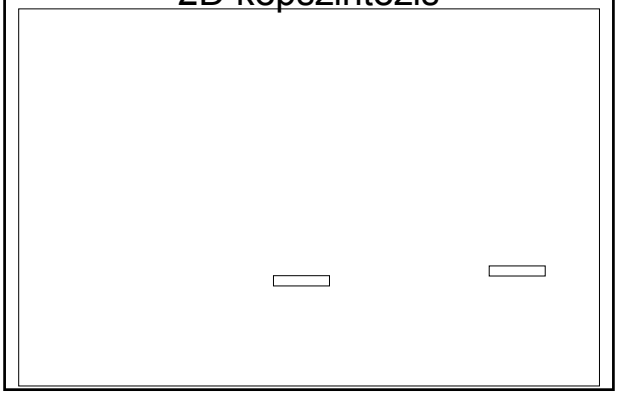
```
Transform {
 translation 0 1 0
 rotation 0 0 1  0.1221
 children [
  DEF antenna Shape {
   geometry Cylinder { radius .05 height 3 }
   appearance Appearance { material Material { diffuseColor 0.7 0.7 0.1 } }
  }
 ]
}

Transform {
 translation 0 1 0
 rotation 0 0 1  -0.1221
 children [ USE antenna ]
}
```



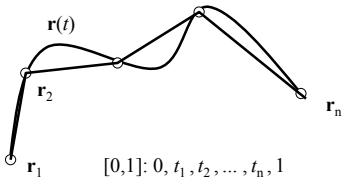
2D képszintézis

Szirmay-Kalos László

2D képszintézis



Vektorizáció



$[0,1]$: $0, t_1, t_2, \ldots, t_n, 1$

$\mathbf{r}_1 = \mathbf{r}(0), \mathbf{r}_2 = \mathbf{r}(t_2), \ldots, \mathbf{r}_n = \mathbf{r}(1)$

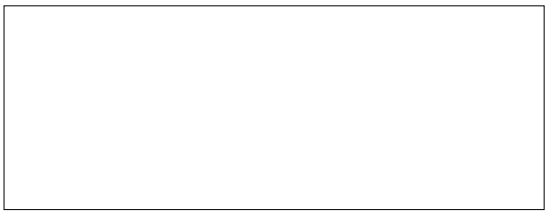
Modellezési transzformáció

$$[x, y] = \mathbf{o} + \alpha\mathbf{u} + \beta\mathbf{v}$$

$$[x, y, 1] = [\alpha, \beta, 1] \begin{bmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ o_x & o_y & 1 \end{bmatrix}$$

$\mathbf{T_M}$

Ablak-nézet transzformáció



$$X = (x - w_x)\, v_w/w_w + v_x$$
$$Y = (y - w_y)\, v_h/w_h + v_y$$

$$[X, Y, 1] = [x, y, 1] \begin{bmatrix} v_w/w_w & 0 & 0 \\ 0 & v_h/w_h & 0 \\ v_x - w_x v_w/w_w & v_y - w_y v_h/w_h & 1 \end{bmatrix}$$

$\mathbf{T_V}$

Összetett transzformáció

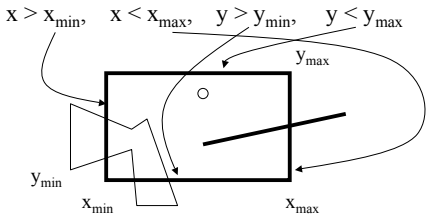
$[X, Y, 1] = ([\alpha, \beta, 1]\ \mathbf{T}_M)\ \mathbf{T}_V$

$[X, Y, 1] = [\alpha, \beta, 1]\ (\mathbf{T}_M\ \mathbf{T}_V) = [\alpha, \beta, 1]\ \mathbf{T}_C$

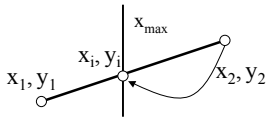
- $\mathbf{T}_V$ nézeti transzformáció számítása
- for each object o
  - $\mathbf{T}_M$ előállítása
  - $\mathbf{T}_C = \mathbf{T}_M\ \mathbf{T}_V$
  - for each point of object o: transzformáció $\mathbf{T}_C$ -vel
- endfor

Vágás

- Vektorizáció miatt: pont, szakasz, poligon
  - Pontok vágása

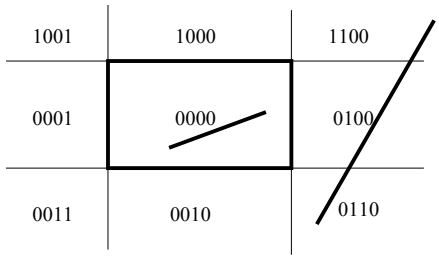


Szakasz vágás félsíkra



- $x(t) = x_1 + (x_2 - x_1)t, \quad y(t) = y_1 + (y_2 - y_1)t$
- $x = x_{max}$
  - Metszéspont:
  - $x_{max} = x_1 + (x_2 - x_1)t \Rightarrow t = (x_{max}-x_1)/(x_2-x_1)$
- $x_i = x_{max} \qquad y_i = y_1 + (y_2 - y_1)\ (x_{max}-x_1)/(x_2-x_1)$

Cohen-Sutherland vágás



Cohen-Sutherland algoritmus

```
BOOL Line2D :: Clip( RectAngle& cliprect ) {
        Point2D& p1 = Point(0);  Point2D& p2 = Point(1);
        int c1 = cliprect.Code( p1 ),  c2 = cliprect.Code( p2 );

        for( ; ; ) {
                if (c1 == 0 && c2 == 0) return TRUE;
                if ( (c1 & c2) != 0 ) return FALSE;
                Metsző határ meghatározás
                Metszéspont számítás
                if (c1 & f) { p1 = pi; c1 = cliprect.Code( pi ); }
                else        { p2 = pi; c2 = cliprect.Code( pi ); }
        }
}
```

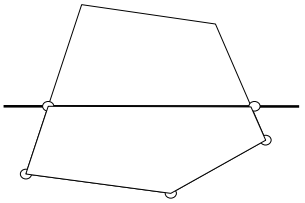
Metsző határ meghatározás

```
int f = 8;
if ( (c1 & 1) != (c2 & 1) )          f = 1;
else if ( (c1 & 2) != (c2 & 2) )     f = 2;
else if ( (c1 & 4) != (c2 & 4) )     f = 4;
```

Metszéspont számítás

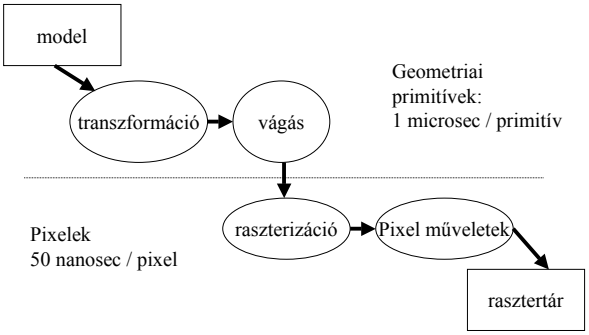
```
double dy = p2.Y() - p1.Y(),   dx = p2.X() - p1.X();

switch ( f ) {
case 1: yi = p1.Y() + dy * (cliprect.Left() - p1.X()) / dx;
        pi = Point2D( cliprect.Left(), yi );      break;
case 2: yi = p1.Y() + dy * (cliprect.Right() - p1.X()) / dx;
        pi = Point2D( cliprect.Right(), yi );    break;
case 4: xi = p1.X() + dx * (cliprect.Bottom() - p1.Y()) / dy;
        pi = Point2D( xi, cliprect.Bottom() ); break;
case 8: xi = p1.X() + dx * (cliprect.Top() - p1.Y()) / dy;
        pi = Point2D( xi, cliprect.Top() );
}
```

Sutherland-Hodgeman poligonvágás

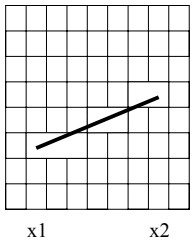


```
PolygonClipping(p[n]⇨ q[m])
    m = 0;
    for( i=0; i < n; i++) {
        if (p[i] belső) {
            q[m++] = p[i]
            if (p[i +1] külső)  q[m++] = Intersect( (p[i],p[i+1]), vágóegyenes)
        } else {
            if  (p[i + 1] külső) q[m++] = Intersect( (p[i],p[i+1]), vágóegyenes)
        }
    }
}
```

Pászta konverzió (raszterizáció)



Szakasz rajzolás



Egyenes egyenlete:

$y = mx + b$

Egyeneshúzás

```
for( x = x1; x <= x2; x++) {
        Y = m*x + b;
        y = Round( Y );
        write( x, y );
}
```

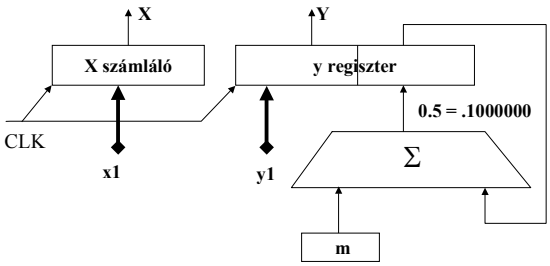
Inkrementális elv

- Y(X) kiszámítása
  - Y(X) = F(Y(X-1)) = Y(X-1) + dY/dX
- Szakasz rajzolás: Y(X) = mX + b
  - Y(X) = F(Y(X-1)) = Y(X-1) + m
- Összeadás: fixpontos ábrázolás: $y = Y\ 2^T$
  - T: hiba < 1 a leghosszabb műveletsornál
  - $N\ 2^{-T} < 1$: $T > \log_2 N$
  - round(y): y+0.5-t csonkítjuk

DDA szakaszrajzolás

```
DDADrawLine(x1, y1, x2, y2) {
        m = (y2 - y1)/(x2 - x1)
        y = y1                              + 0.5
        FOR X = x1 TO x2 {
                Y = round(y)                trunc(y)
                WRITE(X, Y, color)
                y = y+m
        }
}
```

DDA szakaszrajzoló hardver

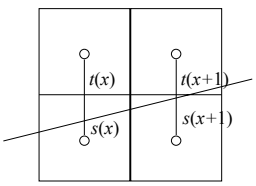


DDA szakaszrajzoló szoftver

```
DDADrawLine(int x1, int y1, int x2, int y2) {
        int m = ((long)(y2 - y1)<<T) / (x2 - x1);
        long y = (long)y1 << T + ((long)1 <<(T-1));
        for( int X = x1; X <= x2; X++) {
                int Y = y >> T;
                Write(X, Y, color);
                y += m;
        }
}
```

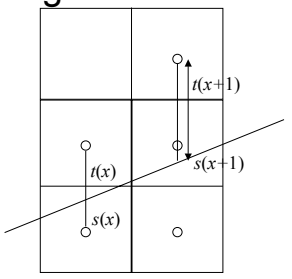
Bresenham algoritmus

$e = s\text{-}t < ??? \; 0$



$s(X+1) = s(X) + Dy/Dx$
$t(X+1) = s(X) - Dy/Dx$
$e(X+1) = e(X) + 2Dy/Dx$

$e(x1) = -1$



$s(X+1) = s(X) + Dy/Dx - 1$
$t(X+1) = s(X) - Dy/Dx + 1$
$e(X+1) = e(X) + 2(Dy/Dx - 1)$

Egész döntési változó

- $e(x1) = -1$
- •/Dx racionális számokkal változik
- e előjele alapján döntünk
- Döntési változó: E = e Dx
- Inkrementális formulák:

$E(X+1) = E(X) + 2Dy$ ha $E < 0$

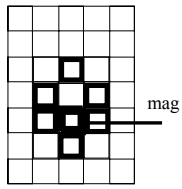
$E(X+1) = E(X) + 2(Dy - Dx)$ ha $E >= 0$

$E(x1) = - Dx$

Bresenham program

```
void BresenhamLine( int x1, int y1, int x2, int y2 ) {
        int dx = x2 - x1, dy = y2 - y1;

        int dep = 2 * (dy - dx), dem = 2 * dy;
        int e = -dx;
        int y = y1;

        for( int x = x1; x <= x2; x++ )
                if (e <= 0)     e += dem;
                else {          e += dep; y++; }
                Write( x, y, color );
        }
}
```

Terület elárasztás működése



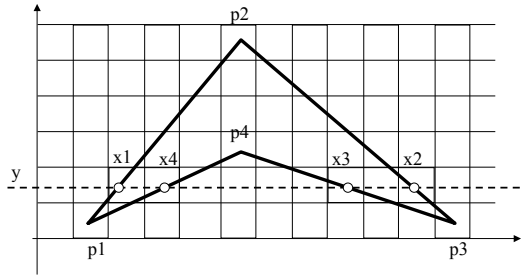

```
FloodFill(x, y) {
        if (pixel[x][y] belső pont) {
                Write(x, y, color);
                Flood(x, y-1);
                Flood(x, y+1);
                Flood(x-1, y);
                Flood(x+1, y);
        }
}
```

Belső pont = sötétkék

Területkitöltés

- Geometriai reprezentáció alapján
  - p1,p2,...,pn csúcsok
  - p1-p2, p2-p3,...pn-p1 élek
- Belső pixel
  - páratlanszor metszünk élt, ha végtelenbôl jövünk
  - végtelen: vágás után a nézet széle

Területkitöltés

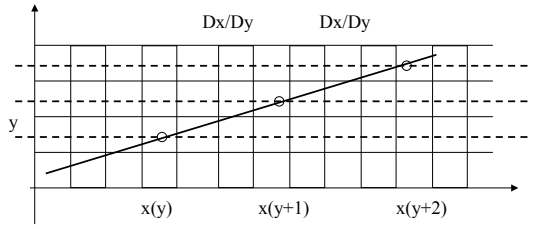


Naív területkitöltés

```
FillPolygonSlow(q[m]) {
    FOR Y=0 TO Ymax DO {
        scanline = Y
        k=0
        FOR e=0 TO m-1 {
            IF scanline a q[e] és q[e+1] csúcsok között van
                x[k++] = (q[e],q[e+1]) szakasz és a scanline metszése
        x[k] tömb rendezése
        FOR i=0 TO k/2 - 1
            FOR X=x[2i] TO x[2i+1] Write(X, Y, Color(X, Y) )
}
```

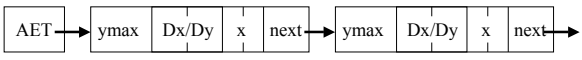
Kitöltés gyorsítása

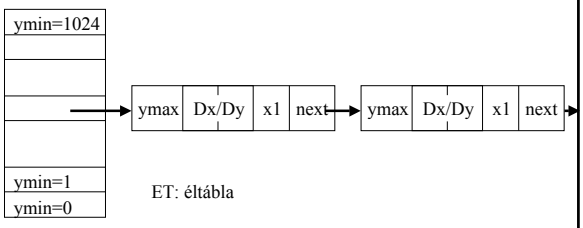
- Vizsgálatok csak az aktív élekre
- Metszéspontszámítás inkrementális elv szerint



Aktív él lista

AET: aktív éltábla





ET: éltábla

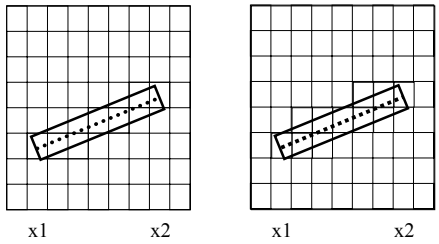
Gyors poligon kitöltő

```
FillPolygon(q[m]) {
    ET felépítése: ET, Ymin, Ymax
    FOR Y=Ymin TO Ymax {
        ET Y. sorát az AET-be másolni
        FOR minden edge élre az AET-ben {
            IF ymax(edge) >= Y  Vedd ki az edge élt az AET-bôl
        AET újra rendezése
        FOR minden második l élre  az AET-ben
            FOR X=round(x[l]) TO round(x[l+1])
                Write(X,Y,Color(X, Y))
        FOR minden l élre az AET-ben x[l] += Dx/Dy
    }
}
```

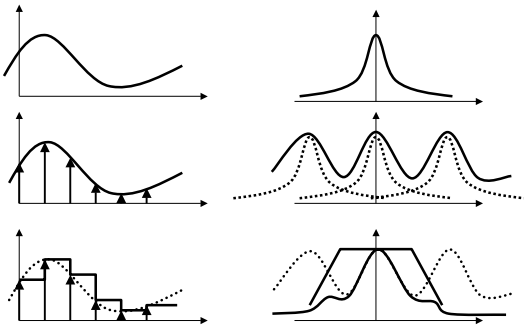
Csipkézettség csökkentés (anti-aliasing)

Szirmay-Kalos László

Csipkézettség csökkentés



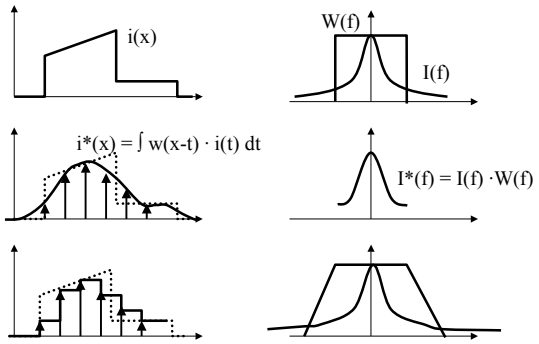
Csipkeszűrés: jelfeldolgozás



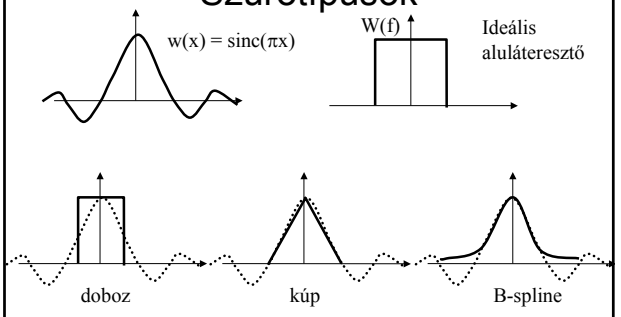
Mintavételi tétel megsértése: Megoldás

- Mintavételi és visszaállítási frekvencia növelése
- Utószűrés:
  - Mintavételi frekvencia növelése, digitális szűrés, alacsonyfrekvenciás visszaállítás
- Előszűrés:
  - „Analóg” szűrés mintavételezés előtt
- Stochastikus mintavételezés

Előszűrés

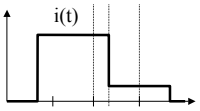


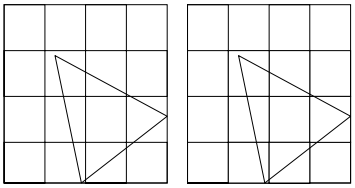
Szűrőtípusok



Doboz szűrő

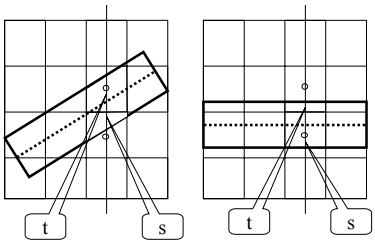
$$i^*(x) = \int w(x-t) \cdot i(t)\, dt = \int_{x-0.5}^{x+0.5} i(t)\, dt$$





$$i^*(x) = \Sigma\, A_j\, i_j$$

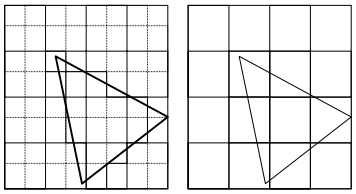
Csipkézetlen szakaszok



$A_s = t$
$A_t = s$

$$A_s(x+1) = A_s(x) + \Delta y/\Delta x$$
$$A_t(x+1) = A_t(x) - \Delta y/\Delta x$$

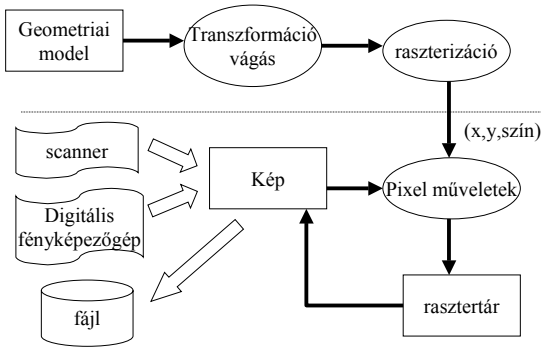
Útószűrés



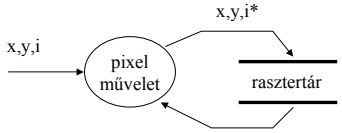
Pixel műveletek, képek

Szirmay-Kalos László

Pixel műveletek



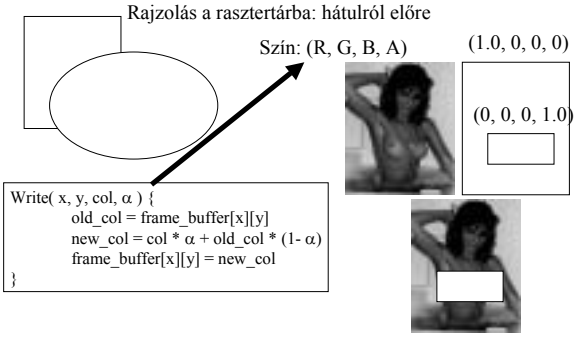
Pixel szintű műveletek



Példák:

- raszteroperációk: XOR, NOT, SET, ADD, AND, …
- átlátszóság
- kvantálás, dither
- bitsík maszkolás
- rétegek
- ollózás

Átlátszóság



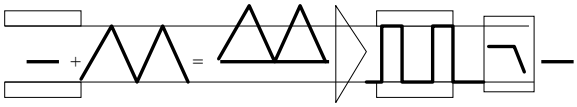
Kvantálási hibák

8 bit: 256 szín

4 bit: 16 szín

3 bit: 8 szín

Dither

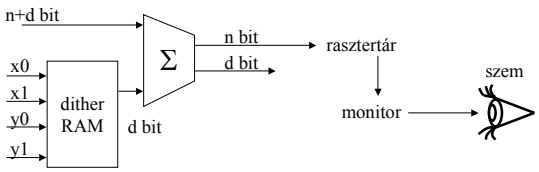


eredeti jel

zaj hozzákeverés

kvantálás

aluláteresztô szűrés



Fekete-fehér dither

Véletlen zaj

halftone

Kis periódusú mátrix dither

$$\frac{1}{16}\begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$$

Nagyperiódusú egyenletes sorozat

Színes dither

Véletlen zaj

Mátrix dither

Képek, képformátumok

- Fej:
  - típus, méret (szélesség, magasság)
  - bit-per-pixel, indexelt-valós szín, lookup tábla
- Törzs:
  - szélesség x magasság db pixel (R,G,B) vagy idx
  - Tömörítés (run length, LZW, Huffmann, FFT, wavelet)
- Standard formátumok:
  - TARGA, GIF, JPEG, TIFF, BMP, PCX
  - GIF, MPG, AVI, ...

TARGA

Fej:

0
0
2
9 db 0
Szélesség alsó bájt
Szélesség felső bájt
Magasság alsó bájt
Magasság felső bájt
Bit-per-pixel: 24
32
Törzs

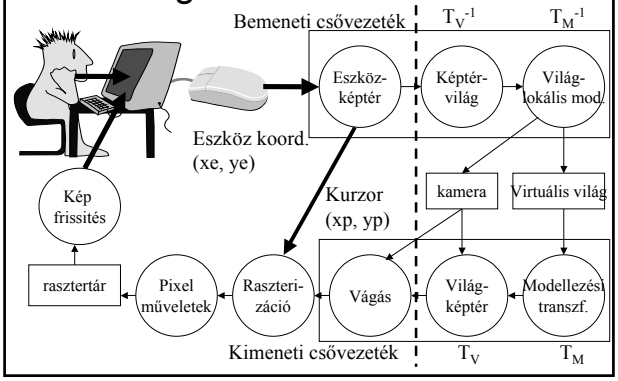
Törzs:

b1
g1
r1
b2
g2
r2
…

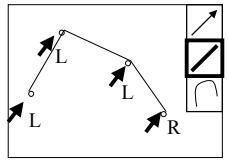
2D grafikus rendszerek

Szirmay-Kalos László

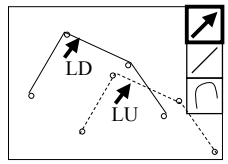
2D grafikus rendszerek



2D grafikus editor: GUI

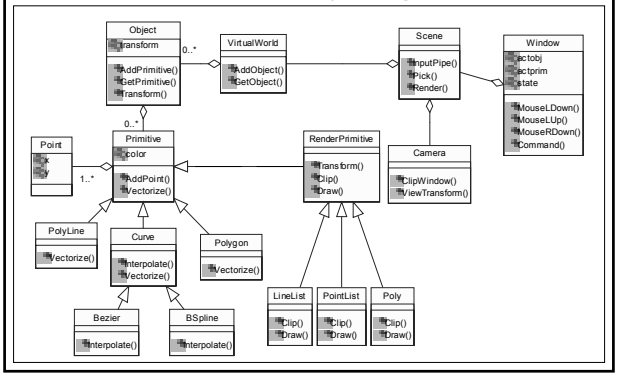


MouseLDown	első pont
MouseLDown	második
...	
MouseLDown	n.
MouseRDown	utolsó



MouseLDown	pick?
MouseMove	mozgat
MouseLUp	letesz

UML osztálydiagram



Kimeneti csővezeték: Render

```
Scene :: Render ( ) {
    Transform Tv = camera.ViewTransform( );
    for each object obj {
        Transform Tm = obj -> Transform( );
        Transform Tc = Tm * Tv;
        for each primitive p of object obj {
            RenderPrimitive * rp = p -> Vectorize( );
            rp -> Transform( Tc );
            if ( rp -> Clip( camera.ClipWindow ) )  rp -> Draw( );
        }
    }
}
```

Vektorizáció

```
class Curve : Primitive {

    virtual Point Interpolate( double tt ) = 0;

    RenderPrimitive Vectorize( ) {
        LineList linelist = new LineList();
        for(int i = 0; i <= NVECTOR; i++)  {
            double t = (double)i / NVECTOR;
            linelist -> AddPoint( Interpolate( t ) );
        }
        return linelist;
    }
};
```

Bezier görbe interpoláció

$$B_i(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

```
class Bezier : Curve {

    Point Interpolate( double tt ) {
        Point rr(0, 0);
        for(int i = 0; i < npoints; i++) {
            double Bi = 1.0;
            for(int j = 1; j <= i; j++)       Bi *= tt * (npoints-j)/j;
            for(       ; j <  npoints; j++)  Bi *= (1-tt);
            rr += points[i] * Bi;
        }
        return rr;
    }
};
```

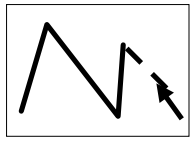
$$\mathbf{r}(t) = \Sigma\, B_i(t)\, \mathbf{r}_i$$

RenderPrimitive

```
class RenderPrimitive {
    Point *     points;
    Color       color;
    void Transform( Transform T ) {
        for each point i do points[i].Transform( T );
    }
    virtual Bool Clip( Rect cliprect ) = 0;
    virtual void Draw( ) = 0;
};

class Line : public RenderPrimitive {
    Line( Point p1, Point p2 ) {
        points = new Point[2]; points[0] = p1; points[1] = p2;
    }
    Bool Clip( Rect cliprect ) { Cohen-Sutherland vágás }
    void Draw( ) { Bresenham algoritmus }
};
```

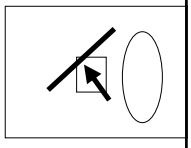
Bemeneti csővezeték: pontok beépítése a virtuális világba

```
Scene :: InputPipeline( X, Y ) {
    Object * obj = world.Object( actobject );
    Transform Tm = obj -> Transform( );
    Transform Tv = camera.ViewTransform( );
    Transfrom Tci = (Tm * Tv).Invert( );
    Point p = Point(X, Y).Transform( Tci );
    world.Object( actobject ) -> Primitive( actprim ) -> AddPoint(p);
}

MouseLDown( X, Y ) {
 ... ha az állapot szerint a pontot be kell építeni
        scene.InputPipeline( X, Y );
        ....
}
```

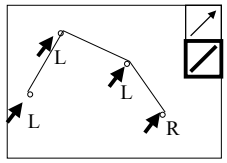


Primitív (objektum) kiválasztása

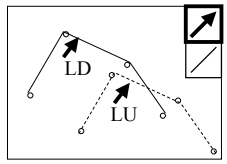
```
Scene :: Pick( X, Y ) {
    Rect pickw( X-5, Y-5, X+5, Y+5 );
    for each object obj {
        Transform Tm = obj -> Transform( );
        Transform Tv = camera.ViewTransform( );
        Transfrom Tc = Tm * Tv;
        for each primitive p of object obj { // prioritás
            RenderPrimitive * rp = p -> Vectorize( );
            rp -> Transform( Tc );
            if ( rp -> Clip( pickw ) ) { actobj = obj; actprim = p; return actobj; }
        }
    }
}

MouseLDown( X, Y ) {
... ha az állapot szerint kiválasztás:   scene.Pick( X, Y );
}
```



Eseményvezérelt program

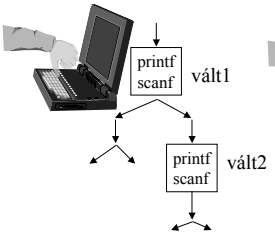


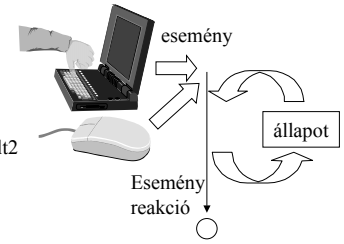
MouseLDown	tárol
MouseLDown	rajzol/tárol
...	
MouseLDown	rajzol/tárol
MouseRDown	rajzol/tárol



MouseLDown	pick?
MouseMove	töröl/rajzol
...	
MouseMove	töröl/rajzol
MouseLUp	rajzol

Interakciós sémák

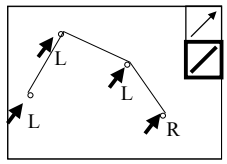


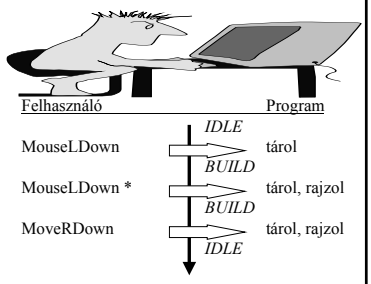


Programvezérelt:
fizikai esemény - változó
összerendelés PC alapján

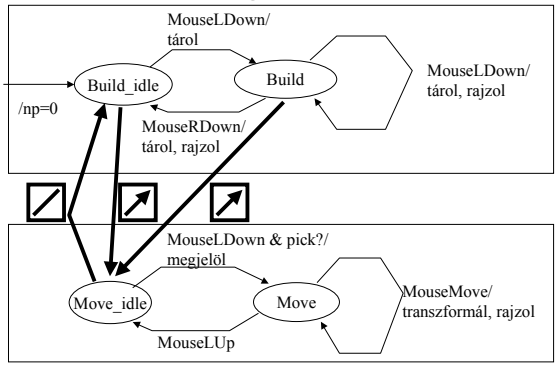
Eseményvezérelt:
összerendelés a belső
állapot alapján

Interakciós sémák tervezése

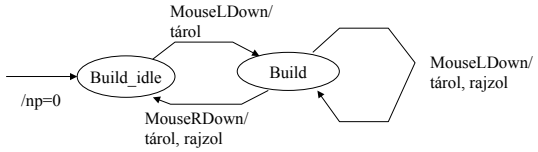




Állapotgép modell

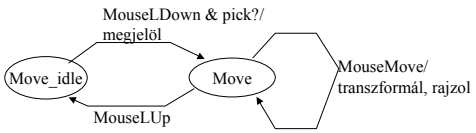


Állapotgép modell: Build



	MouseLDown	MouseRDown
Build_idle	x[np]=xinp, y[np++]=yinp state = Build	
Build	x[np]=xinp, y[np++]=yinp ReDraw( ); state = Build	x[np]=xinp, y[np++]=yinp ReDraw(); state = Build_idle

Állapotgép modell: Move



	MouseLDown	MouseMove	MouseLUp

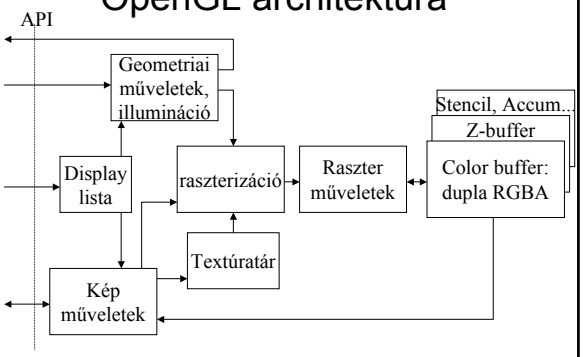
Move_idle	Obj = Pick(xinp, yinp)
	if (Obj != NULL) {
	Obj -> ChangeStyle( )
	xp = xinp, yp = yinp
	state = Move
	}		

Move		Obj->ConcTransf(xinp-xp, yinp-yp) xp = xinp, yp = yinp ReDraw();	state = Move_idle

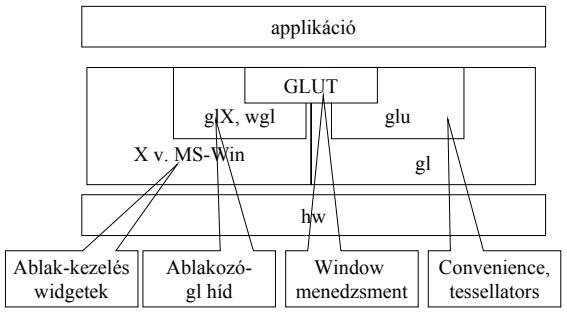
OpenGL

- Képszintézis könyvtár (2D, 3D), API
  - 2D, 3D geometria
  - rajzolási állapot (state)
  - rasztermüveletek
- Op. Rendszer, Ablakozó rendszer független
  - ő nem ablakoz

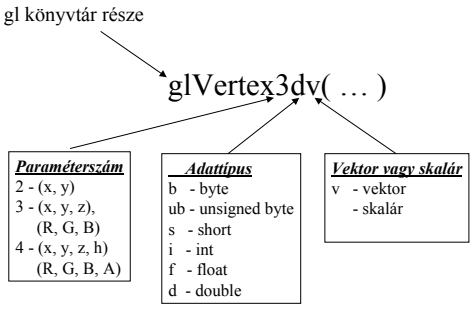
OpenGL architektúra



OpenGL elhelyezkedése



OpenGL szintaxis



OpenGL alkalmazás (GLUT)

- Ablak megnyitás
- Rajzolási állapot, display lista inicializálás
- Display callback:
  - képernyő törlés, állapotváltás, rajzolás, buffercsere
- Reshape callback:
  - nézeti transzformácó, vágás átállítása
- Input callback (mouse, keyboard)
- Idle callback (animáció)
- Üzenethurok

OpenGL: inicializálás

```
main(argc, argv) {
   glutInitWindowSize(200, 200);
   glutInitWindowPosition(100, 100);
   glutInit(&argc, argv);
   glutInitDisplayMode( GLUT_RGBA );

   glutCreateWindow("Sample Window");

   glutMouseFunc( MousePress );     // callback függvények
   glutMotionFunc( MouseMotion );
   glutDisplayFunc( ReDraw );

   glutMainLoop();                  // fő hurok
}
```

OpenGL: eseménykezelés

```
void ReDraw( ) {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   window.scene.Render( );
}

void MousePress( int button, int state, int x, int y ) {
    if (button == GLUT_LEFT &&
        state == GLUT_DOWN)     window.MouseLDown(x, y);
   …
}

void MouseMotion( int x, int y ) {
   window.MouseMove(x, y);
}
```

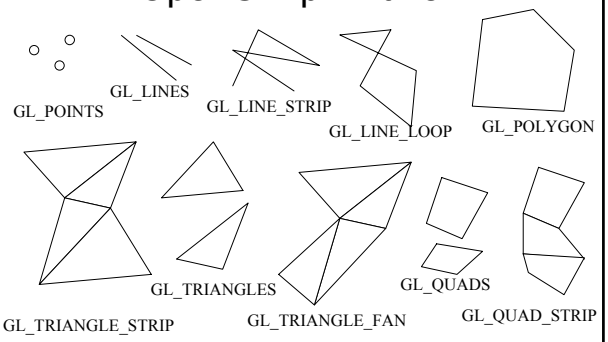
OpenGL Render

```
Scene :: Render ( ) {
    glViewport( „camera.Viewport” );

    for each object obj {
        glLoadIdentity();
        gluOrtho2D( „camera.Window” );
        glMulMatrix( obj->Transform( ) );

        for each primitive p of object obj {
            RenderPrimitive * rp = p -> Vectorize( );
            rp -> Draw( );
        }
    }
}
```

OpenGL LineList

```
class LineList {

    void Draw( ) {
        glColor3d( color.R, color.G, color.B );

        glBegin(GL_LINE_STRIP);
        for( i = 0; i < npoints; i++ )
            glVertex2d(points[i].x, points[i].y);
        glEnd( );

        glFlush( );
    }
};
```

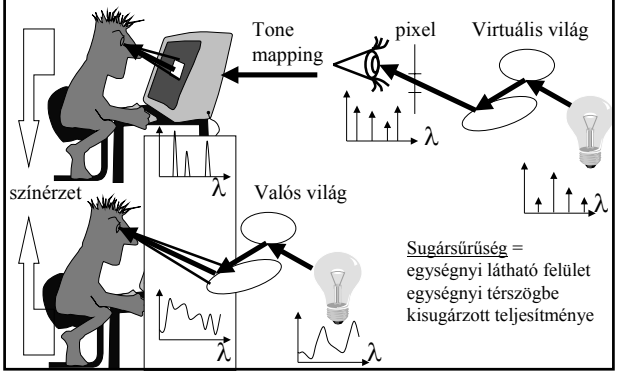
OpenGL: primitívek



3D képszintézis fizikai alapmodellje

László Szirmay-Kalos

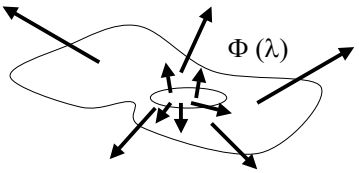
Képszintézis = valós világ illuziója



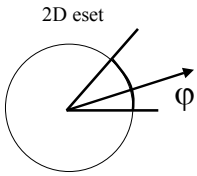
Sugársűrűség = egységnyi látható felület egységnyi térszögbe kisugárzott teljesítménye

Fényerősség mértékei: Fluxus

- Egy felületen átlépő teljesítmény [Watt]
- Fotonok száma
- Spektrum: $\Phi\ [\lambda, \lambda+d\lambda]$



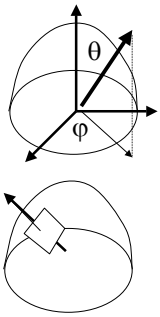
Irányok, irány halmazok: 2D



Irány:
φ szög
egy referencia iránytól

Irány halmaz:
szög [rad]
egység kör íve
Méret: ívhossz
Teljes halmaz: 2π

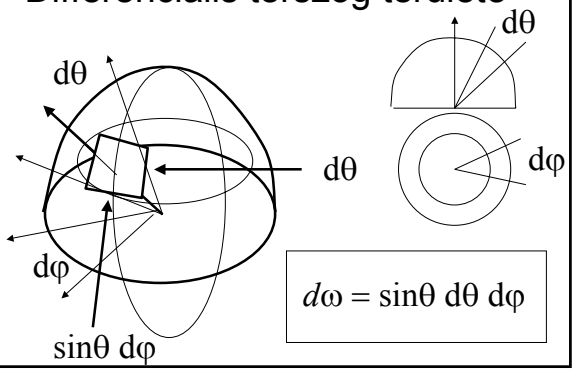
Irányok, irány halmazok : 3D



Irány:
θ,φ szögek
2 referencia iránytól

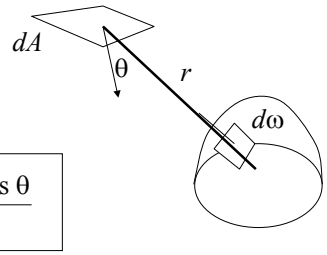
Irány halmaz:
térszög [sr]
tartomány az egységgömbön
Méret: tartomány területe
Teljes halmaz: 4π

Differenciális térszög területe



$$d\omega = \sin\theta \, d\theta \, d\varphi$$

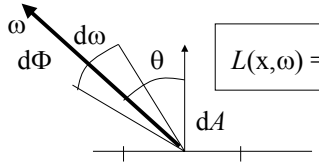
Térszög, amelyben egy differenciális felület látható



$$d\omega = \frac{dA \cos\theta}{r^2}$$

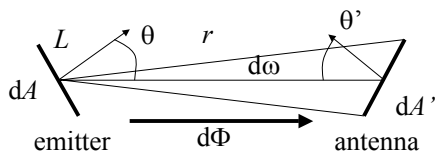
Radiancia: $L(x,\omega)$

1 Egy egységnyi látható felület által egységnyi térszög alatt kibocsátott teljesítmény
[Watt/ sr/ m^2]



$$L(x,\omega) = d\Phi / (dA \cos\theta \, d\omega)$$

Fényátadás két differenciális felületelem között

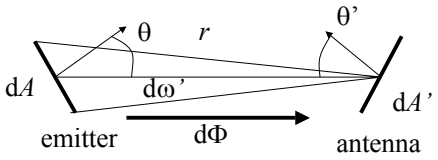


$$d\Phi = L \, dA \cos\theta \, d\omega = L \, \frac{dA \cos\theta \, dA' \cos\theta'}{r^2}$$

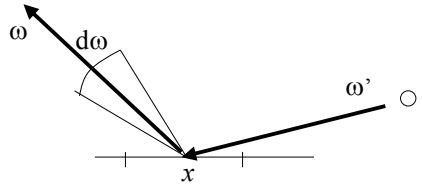
A forrás és az antenna szimmetriája

$$d\Phi = L \frac{dA \cos\theta \; dA' \cos\theta'}{r^2} = L \, dA' \cos\theta' \, d\omega'$$

$d\omega'$



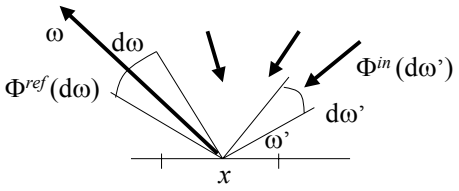
Fény-felület kölcsönhatás



Visszaverődés valószínűség sűrűségfüggvénye:

$\boldsymbol{w}(\omega', x, \omega)\, d\omega$ = Pr{foton $\omega \pm d\omega$ -ba megy | ω'-ből jön}

Adott térszögbe visszavert fluxus



$$\Phi^{ref}(d\omega) = \Phi^{e}(d\omega) + \int_{\Omega} \Phi^{in}(d\omega') \, \boldsymbol{w}(\omega', x, \omega) \, d\omega$$

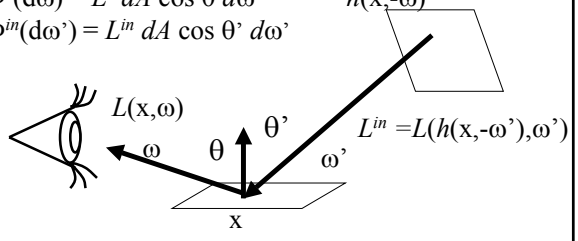
Visszavert radiancia

$\Phi^{ref}(d\omega) = L \, dA \cos\theta \, d\omega$

$\Phi^{e}(d\omega) = L^{e} \, dA \cos\theta \, d\omega$

$\Phi^{in}(d\omega') = L^{in} \, dA \cos\theta' \, d\omega'$

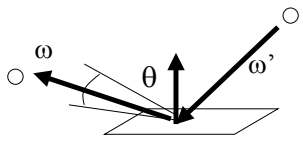
Láthatóság függvény $h(x,-\omega)$



Helyettesítés és osztás

$dA \cos\theta \, d\omega$ -val

$$L(x,\omega) = L^{e}(x,\omega) + \int_{\Omega} L(h(x,-\omega'),\omega') \frac{w(\omega',x,\omega)}{\cos\theta} \cos\theta' d\omega'$$



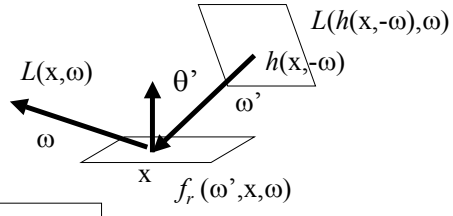
$$\frac{w(\omega',x,\omega)}{\cos\theta} = f_r(\omega',x,\omega)$$

Bidirectional Reflectance Distribution Function

BRDF: $f_r(\omega',x,\omega)$ [1/sr]

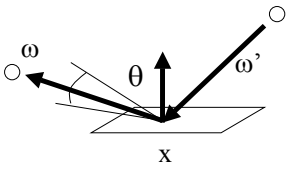
Rendering egyenlet

$$L(x,\omega) = L^{e}(x,\omega) + \int_{\Omega} L(h(x,-\omega'),\omega') \, f_r(\omega',x,\omega) \cos\theta' d\omega'$$



$$L = L^{e} + \tau \, L$$

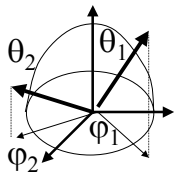
BRDF definició



$$\frac{w(\omega',x,\omega)}{\cos\theta} = f_r(\omega',x,\omega)$$

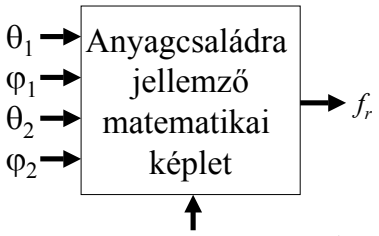
Bidirectional Reflectance Distribution Function
BRDF: $f_r(\omega',x,\omega)$ [1/sr]

Mért BRDF adatok reprezentálása



BRDF 5-változós függvény:
$\theta_1, \varphi_1, \theta_2, \varphi_2, \lambda$
Tábla méret: 100x100x100x100x10 = 10^9

Matematikai BRDF modellek



BRDF tulajdonságok

- 1. Pozitív
- 2. Szimmetrikus (reciprok) - Helmholtz

$$f_r(\omega',x,\omega) = f_r(\omega,x,\omega')$$

- 3. Energia megmaradást nem sértő:
  - a visszavert energia kisebb mint a beérkező
  - a visszaverési valószínűség 1-nél kisebb

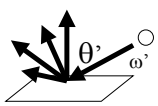
Albedo

- Annak valószínűsége, hogy a ω' irányból érkező foton visszaverődik (valahova):

$$a(x,\omega') = \int w(\omega',x,\omega)\, d\omega = \int f_r(\omega',x,\omega)\cos\theta\, d\omega$$

- Energia megmaradás:

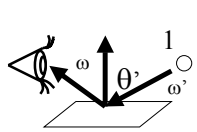
$$a(x,\omega') < 1$$



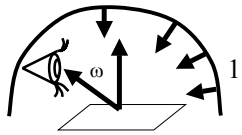
Albedo láthatósága

- Homogén égboltfény

$$a(x,\omega) = \tau 1 = \int f_r(\omega',x,\omega)\cos\theta' d\omega'$$



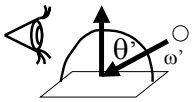
$L^{ref} = 1\, f_r(\omega',x,\omega)\cos\theta'$



$L^{ref} = 1\, a(x,\omega)$

Diffúz visszaverődés

- Radiancia a nézeti iránytól független

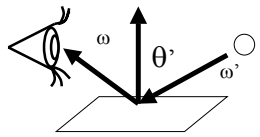


- Helmholtz: a megvilágítástól is független
- A BRDF konstans:

$$f_r(\omega', x, \omega) = k_d(\lambda)$$

Lambert törvény

- Pont fényforrásra válasz



$$L^{ref} = L^{ir}\ k_d \cos\theta'$$

```
class Diffuse {
   Color Kd;
public:
   Color BRDF(Vec& L, Vec& N, Vec& V) { return Kd; }
   Color Albedo( Vec& N, Vec& V ) { return Kd*M_PI; }
};
```

Diffúz objektumok

A diffúz BRDF fizikailag plauzibilis

$$f_r(\omega', x, \omega) = k_d$$

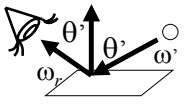
- Pozitív: ✓
- Szimetrikus: ✓
- Energia megmaradás: $k_d < 1/\pi$

$$a(x,\omega') = \int k_d \cos\theta\, d\omega = \int_\phi \int_\theta k_d \cos\theta \sin\theta\, d\theta d\phi = k_d\, 2\pi \int_\theta \cos\theta \sin\theta\, d\theta = k_d\, \pi$$

($d\omega$)

Ideális visszaverődés

- Visszaverődés a tükörirányba

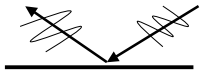


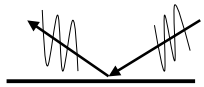
$$L^{ref} = L^{ir}\ k_r$$

- A BRDF Dirac-delta:

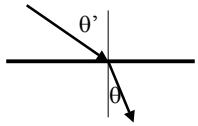
$$f_r(\omega', x, \omega) = \delta(\omega - \omega_r)\, k_r / \cos\theta'$$

Ideális tükrök visszaverése: k_r - Fresnel függvény





$$F_{\parallel} = \left| \frac{\cos\theta - (n + kj)\cos\theta'}{\cos\theta + (n + kj)\cos\theta'} \right|^2 \qquad F_{\perp} = \left| \frac{\cos\theta' - (n + kj)\cos\theta}{\cos\theta' + (n + kj)\cos\theta} \right|^2$$

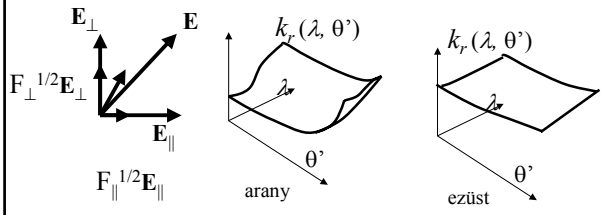


$$n = \frac{\sin\theta'}{\sin\theta}$$

Snellius-Descartes törési törvény
n = a hullám relatív sebessége

Ha a fény nem polarizált

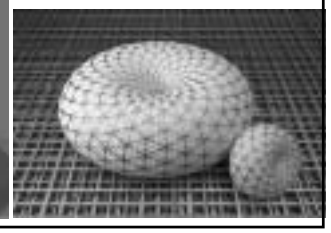
$$k_r(\lambda, \theta') = \frac{\left|F_{\parallel}^{1/2}\,\mathbf{E}_{\parallel} + F_{\perp}^{1/2}\,\mathbf{E}_{\perp}\right|^2}{\left|\mathbf{E}_{\parallel} + \mathbf{E}_{\perp}\right|^2} = \frac{F_{\parallel} + F_{\perp}}{2}$$



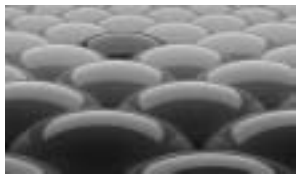
arany

ezüst

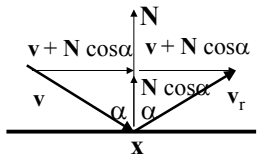
Fémek: törésmutató komplex



Nem fémek: törésmutató valós



Tükörirány számítása



$\cos\alpha = -(\mathbf{v}\cdot\mathbf{N})$

$\mathbf{v}_r = \mathbf{v} + 2\cos\alpha\,\mathbf{N}$

L = $\mathbf{v}_r$, V= **v**

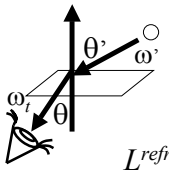
```
ReflectDir( L, N, V ) {
    L = V - N * (N * V) * 2;
}
```

Ideális törés

- A radiancia csak a törési irányba

$$n = \frac{\sin\theta'}{\sin\theta}$$

Snellius-Descartes törési törvény

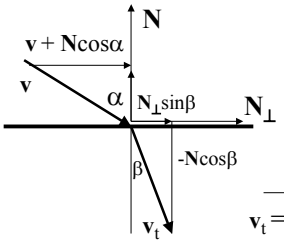


$$L^{refract} = L^{ir}\ k_t$$

- A BRDF Dirac-delta függvény:

$$f_r(\omega', x, \omega) = \delta(\omega - \omega_t)\,k_t/\cos\theta'$$

Törési irány



$$n = \frac{\sin\alpha}{\sin\beta}$$ Snellius-Descartes

$$\mathbf{N}_{\perp} = \frac{\mathbf{v} + \mathbf{N}\cos\alpha}{\sin\alpha}$$

$$\mathbf{v}_t = \mathbf{N}_{\perp}\sin\beta - \mathbf{N}\cos\beta$$

$$\mathbf{v}_t = \mathbf{v}/n + \mathbf{N}(\cos\alpha/n - \cos\beta)$$

$$\cos\beta = \sqrt{1-\sin^2\beta} = \sqrt{1-\sin^2\alpha/n^2}$$

$$\mathbf{v}_t = \mathbf{v}/n + \mathbf{N}\left(\cos\alpha/n - \sqrt{(1-(1-\cos^2\alpha)/n^2)}\right)$$

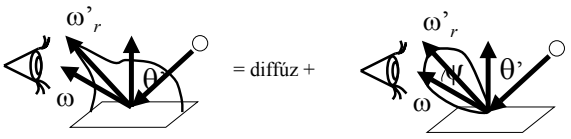
Fénytörő anyagok

Ideális törő

$L = \omega_t$, $V = \omega$

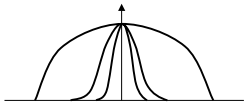
```
class Refractor {
   Color  Kt;
   double n;
public:
   BOOL RefractionDir(Vec& L, Vec& N, Vec& V, BOOL out) {
      double cn = n;
      if ( !out ) cn = 1.0/cn;
      double cosa = N * V;      // Snellius-Descartes law
      double disc = 1 - (1 - cosa * cosa) / cn / cn;
      if (disc < 0) return FALSE;
      L = N * (cosa / cn - sqrt(disc)) - V / cn;
      return TRUE;
   }
};
```

Spekuláris visszaverődés: Phong modell



Kell egy függvény, ami nagy ψ=0 -ra és gyorsan csökken

$k_s \cos^n \psi$

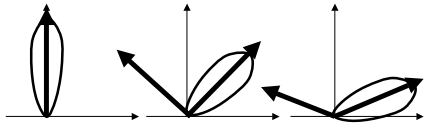


Eredeti Phong modell

$L^{ref} = L^{ir} k_s \cos^n \psi$

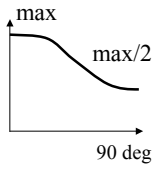
$f_r(\omega', x, \omega) = k_s \cos^n \psi / \cos\theta'$

1 Nem szimmetrikus!

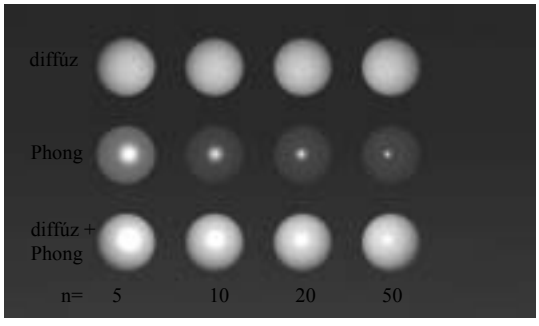


Visszaverődési hurkák

"Albedo": Visszaverődés valószínűsége



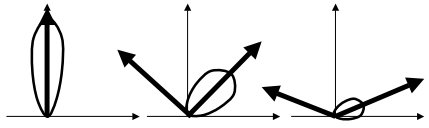
Diffúz+Phong anyagok



Reciprok Phong modell

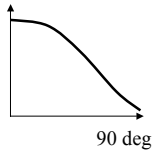
$f_r(\omega', x, \omega) = k_s \cos^n \psi$

$L^{ref} = L^{ir} k_s \cos^n \psi \cos\theta'$

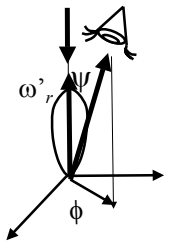


Visszaverődési hurkák
Sötét nagy szögekre

Albedo:



Reciprok Phong albedoja
Merőleges megvilágítás



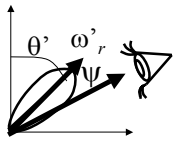
$\psi = \theta$!

$a(x,\omega') = \int k_s \cos^n\psi \cos\theta \, d\omega =$

$\int_\phi \int_\psi k_s \cos^{n+1}\psi \sin\psi \, d\psi \, d\phi =$

$a_{max}(x,\omega') = 2\pi k_s/(n+2)$

Reciprok Phong albedoja
Tetszőleges irány, fényes felület



θ kb. konstans a hurkában
és megegyezik θ'-val

$a(x,\omega') = \int k_s \cos^n\psi \cos\theta \, d\omega =$

$\cos\theta' \int k_s \cos^n\psi \, d\omega =$

$a(x,\omega') \approx 2\pi k_s/(n+1) \cos\theta' \cdot \text{Cutting}(1..0.5)$

Reciprok Phong BRDF class

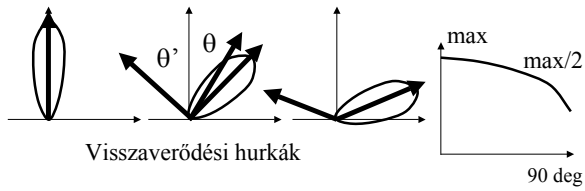
```
class Phong {
   Color  Ks;
   double shine;
public:
   Color BRDF(Vec& L, Vec& N, Vec& V) {
      double cos_in = L * N;
      if (cos_in > 0 && ks() != 0) {
         Vec R = N * (2.0 * cos_in) - L;
         double cos_refl_out = R * V;
         if (cos_refl_out > 0)  return (Ks * pow(cos_refl_out, shine));
      }
      return SColor(0);
   }
   Color Albedo(Vec& N, Vec& V)
         { return ks()*2*M_PI/(shine+2) * (N*V); }
};
```

Max Phong modell

$f_r(\omega',x,\omega) = k_s \cos^n\psi / \max(\cos\theta',\cos\theta)$

$L^{ref} = L^{ir} k_s \cos^n\psi$ if $\theta' < \theta$

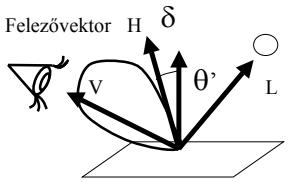
$L^{ref} = L^{ir} k_s \cos^n\psi \cos\theta'/\cos\theta$ if $\theta' > \theta$



Visszaverődési hurkák

Reciprok és max Phong

Phong-Blinn modell (OpenGL)



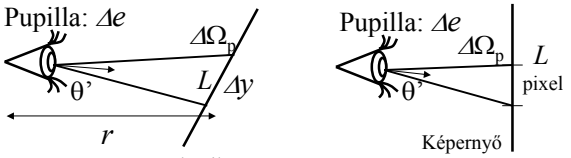
H = (L+V)/|L+V|

$k_s \cos^n \delta$

$L^{ref} = L^{ir} k_s \cos^n \delta$

$f_r(\omega',x,\omega) = k_s \cos^n \delta / \cos\theta'$

A radiancia mérése

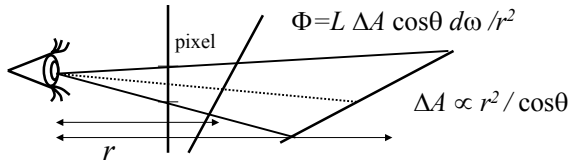


$$\Phi \approx L\, \Delta e \cos\theta'\, \Delta\Omega_p$$

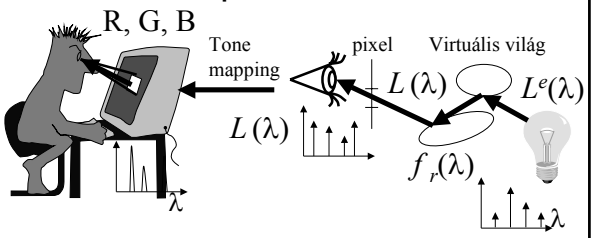
Akkor látszik ugyanolyan fényesnek, ha a radiancia megegyezik
Független a távolságtól és az orientációs szögtől

Miért a radiancia?

A pixel szín a radianciával arányos és független
a felület távolságától és orientációjától



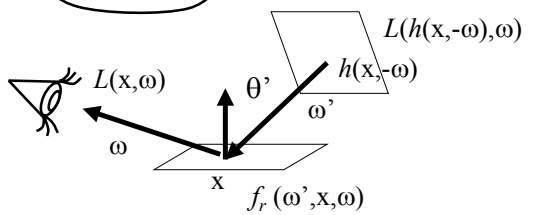
Képszintézis



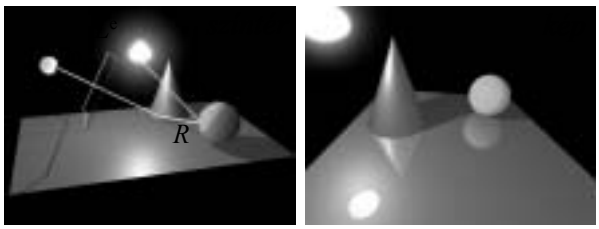
- Pixelben látható felület meghatározása (takarás)
- A látható pont szem irányú sugársűrűsége (árnyalás)
- Skálázás, színleképzés (Tone mapping)

Csatolás az árnyalási egyenletben

$$L(x,\omega) = L^e(x,\omega) + \int L(h(x,-\omega),\omega)\, f_r(\omega', x, \omega) \cos\theta'\, d\omega'$$

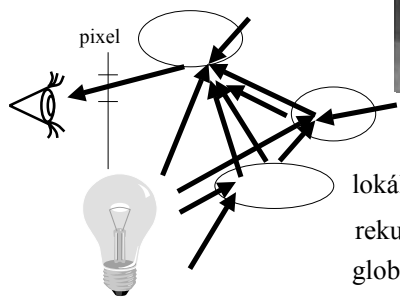


Számítógépes képszintézis



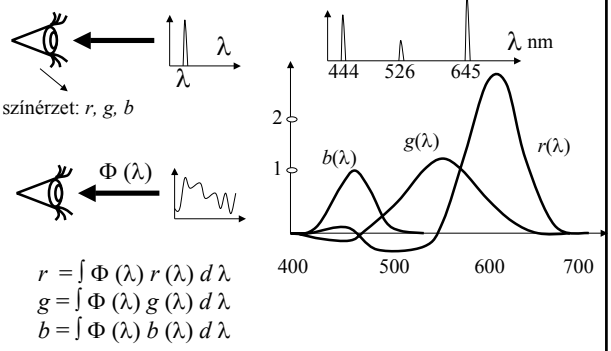
képszintézis = integrál a fényutak terében

Megoldási kompromisszumok



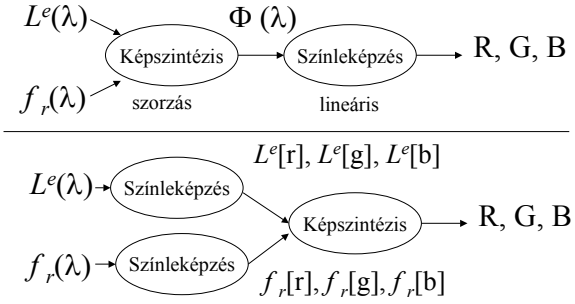
lokális illumináció
rekurzív sugárkövetés
globális illumináció

Színleképzés: Tone mapping



$r = \int \Phi(\lambda)\, r(\lambda)\, d\lambda$
$g = \int \Phi(\lambda)\, g(\lambda)\, d\lambda$
$b = \int \Phi(\lambda)\, b(\lambda)\, d\lambda$

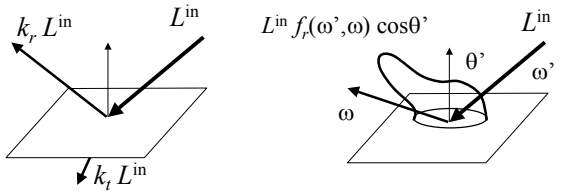
Spektrális versus RGB képszintézis



Sugárkövetés: ray-casting, ray-tracing

Szirmay-Kalos László

Fény-Felület kölcsönhatás



Koherens tagok
geometriai optika

Inkoherens tagok
valószínűség számítás

BRDF: $f_r(\omega',\omega) = L^{out}/L^{in} \cos\theta'$
kétirányú visszaverődés függvény

Lokális illuminációs módszer

$$L(x,\omega)=L^e(x,\omega)+\int L(h(x,-\omega),\omega)\, f_r(\omega',x,\omega) \cos\theta' d\omega'$$

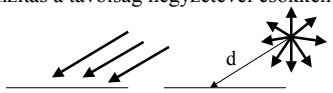
- A jobb oldali radiancia:
  - fényforrások emissziója
- Fényforrások fényének egyszeri visszaverődését számítjuk

$$L(x,\omega)=L^e(x,\omega)+\int L_{lightsource} f_r(\omega',x,\omega) \cos\theta' d\omega'$$

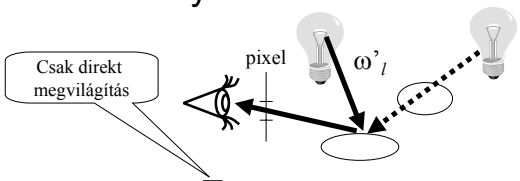
$$\approx L^e(x,\omega)$$

Absztrakt fényforrás modellek: az integrálást megspórolják

- **Ambiens fényforrás**: L_a constant
  - ambiens visszaverődési modell: $L^{ref} = L_a k_a$
  - égbolt fény modell: $L^{ref} = L_a a(x,$ view dir)
- **Irány fényforrások**: egyetlen irányba sugároz, a fénysugarak párhuzamosak, az intenzitás független a pozíciótól
- **Pozicionális fényforrás**: egyetlen pontból sugároz, az intenzitás a távolság négyzetével csökken



Lokális illumináció absztrakt fényforrásokkal



$$L(\omega) \approx L^e(\omega) + \sum_l r_l \, L_l(\omega'_l) f_r(\omega'_l, \omega) \cos\theta'_l$$

0/1: Fényforrás láthatóság

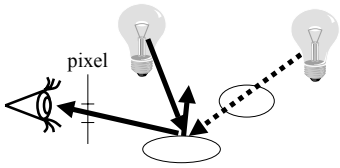
Pont, irányfényforrásokból származó megvilágítás

Ambiens tag

Lokális illumináció + ambiens tag

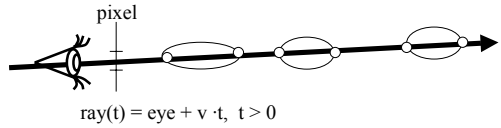
$$L(\omega) \approx L^e(\omega) + \sum_l r_l \, L_l(\omega'_l) f_r(\omega'_l, \omega) \cos\theta'_l + k_a L_a$$

Lokális illuminációs algoritmusok



- Láthatóság számítás a szemből
- Fényforrás láthatóság számítás a látható pontból
- A fényforrás fényének visszaverése a nézeti irányba: felületi normális

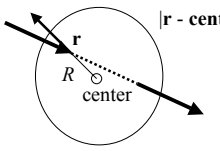
Láthatóság a szemből



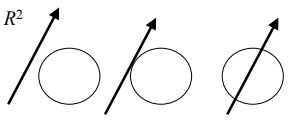
ray(t) = eye + v ·t, t > 0

```
double FirstIntersect(ray ⇨ iobject, x)          // < 0 if no intersection
   t = -1
   FOR each object
        tnew = Intersect( ray, object );   // < 0 if no intersection
        IF (tnew > 0 && (tnew < t || t <0) )  t = tnew, iobject = object
   ENDFOR
   IF (t > 0)  x = eye + v ·t;
   RETURN t;
END
```

Metszéspont számítás gömbbel



$|\mathbf{r} - \mathbf{center}|^2 = R^2$



$|\mathbf{ray}(t) - \mathbf{center}|^2 = R^2$

$(\mathbf{v} \cdot \mathbf{v})\, t^2 + 2\,((\mathbf{eye\text{-}center}) \cdot \mathbf{v})\, t + ((\mathbf{eye\text{-}center}) \cdot (\mathbf{eye\text{-}center})) - R^2 = 0$

Wanted: minimum pozitív megoldás

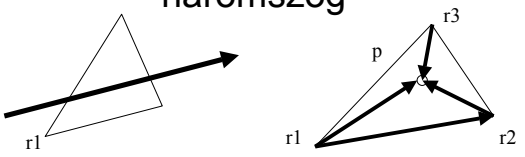
Felületi normális: $(\mathbf{ray}(t) - \mathbf{center})/R$

Kvadratikus felületek

Kvadratikus felület: $[x,y,z,1]\,\mathbf{A}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$ ⇒ Másodfokú egyenlet

Ellipszoid	Végtelen kúp	Végtelen henger
$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}-1=0$	$\frac{x^2}{a^2}+\frac{y^2}{b^2}-z^2=0$	$\frac{x^2}{a^2}+\frac{y^2}{b^2}-1=0$

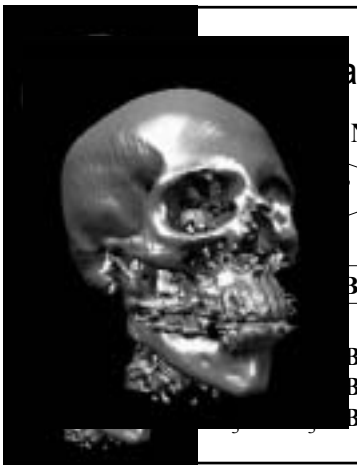
Metszéspont számítás: háromszög



1. Síkmetszés: $(ray(t) - r1) \cdot n = 0, \quad t > 0$
   normál: $\mathbf{n} = (\mathbf{r2} - \mathbf{r1}) \times (\mathbf{r3} - \mathbf{r1})$

2. A metszéspont a háromszögön belül van-e?
   $((r2 - r1) \times (p - r1)) \cdot n > 0$
   $((r3 - r2) \times (p - r2)) \cdot n > 0$
   $((r1 - r3) \times (p - r3)) \cdot n > 0$

Felületi normális: n
vagy „shading normals”



aló normálok

$\mathbf{N}$ $\mathbf{N}_1$ (X_1, Y_1, Z_1)

$\mathbf{B}\,Y + \mathbf{C}$ ⇐ **A**, **B**, **C**

$\mathbf{B}\,Y_1 + \mathbf{C}$
$\mathbf{B}\,Y_2 + \mathbf{C}$
$\mathbf{B}\,Y_3 + \mathbf{C}$
} 3 változós lineáris egyenletrendsze

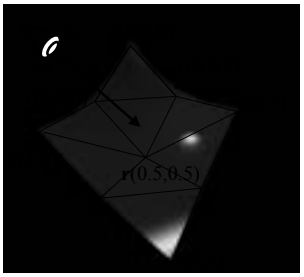
Parametrikus felületek

$\mathbf{r}(u,v)$, u,v in $[0,1]$
$\mathbf{ray}(t) = \mathbf{eye} + \mathbf{v} \cdot t, \quad t > 0$

$\mathbf{r}(u,v) = \mathbf{ray}(t)$

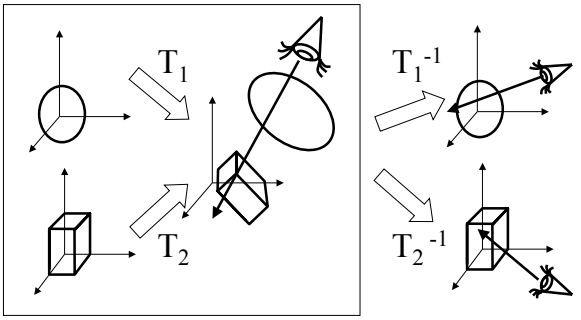
Egyenlet megoldás:
u,v,t

Teszt:
$0 < u,v < 1$,
$t > 0$



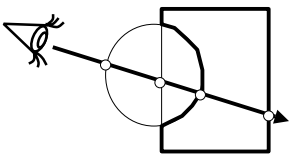
Rekurzív tesszelláció

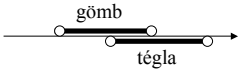
Transzformált objektumok



CSG modellek

Különbség:



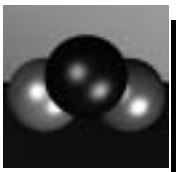


Reguláris halmazművelet a szakaszokra:

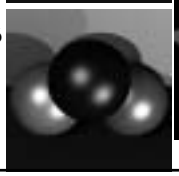
Ray-casting képek

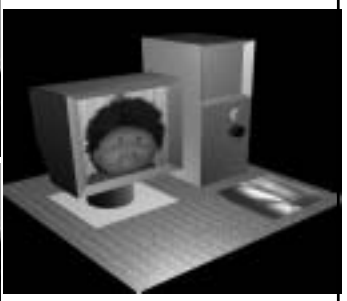
Lokális illumináció

valós idő

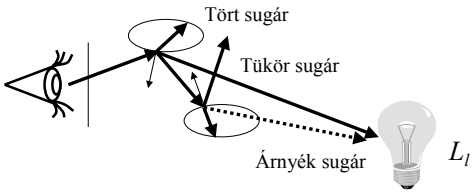


Lokális illumináció + árnyékok 0.1 sec … 1 sec





Sugárkövetés



$$L(\omega) = L^e(\omega) + \sum_l r_l \, L_l(\omega'_l) f_r(\omega'_l, \omega) \cos\theta'_l + k_a L_a + k_r L^{in}(\omega_r) + k_t L^{in}(\omega_t)$$

Tükör irányból érkező fény

Törési irányból érkező fény

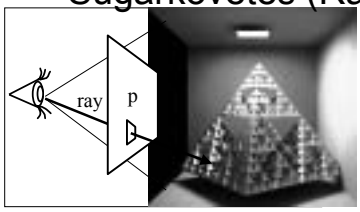
Sugárkövetés lépései

1 A szembe a pixel irányából érkező sugársűrűség
- Adott irányba látható felületi pont és normálvektor
- A felületi pontból látható fényforrások
- A felületi pontban a tükör és törési irány
  - A tükör irányból érkező sugársűrűség
  - A törési irányból érkező sugársűrűség

Rekurzió

- Az árnyalási egyenlet kiértékelése

Sugárkövetés (Ray-tracing)

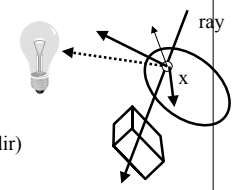


p RGB

Render( )
  for each pixel p
    Ray r = ray(eye ⇨ pixel p)
    color = **Trace**(ray)
    RGB = ToneMapping(color)
    WritePixel(p, RGB)
  endfor
end

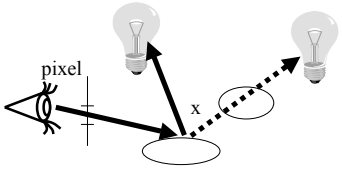
Sugárkövetés: Trace függvény

Color Trace(ray, d)
  IF d > dmax THEN RETURN L_a
  IF (!FirstIntersect(ray ⇨ object, x))
    RETURN L_a
  ENDIF
  color = L^e (x, -ray.dir)
  color += Direct Lightsource(x, -ray.dir)
  IF ($k_r > 0$) THEN
    ReflectDir(ray, reflected ray)
    color += $k_r \cdot$ Trace(reflected ray**, d+1**)
  ENDIF
  IF ($k_t > 0$ && RefractDir(ray, refracted ray))
    color += $k_t \cdot$ Trace(refracted ray**, d+1**)
  ENDIF
RETURN color

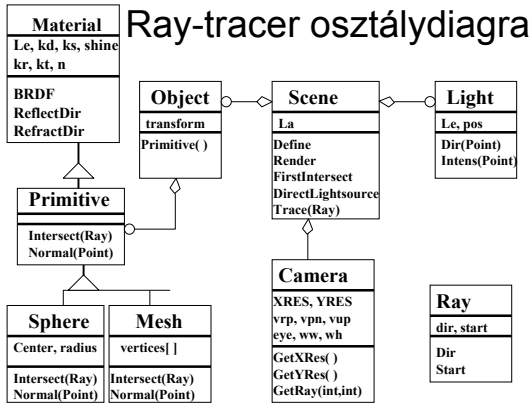


DirectLightsource

DirectLightsource(x, vdir)
  color = $k_a L_a$
  FOR each lightsource l DO árnyék
    shadowray = x to lightsource[l]
    t = FirstIntersect(shadowray);
    IF (t < 0 || t > |x - lightsource[l].pos|)
      color += Brdf(ldir, x, vdir) cos θ_l' lightsource[l].Intensity
    ENDIF
  ENDFOR
RETURN color



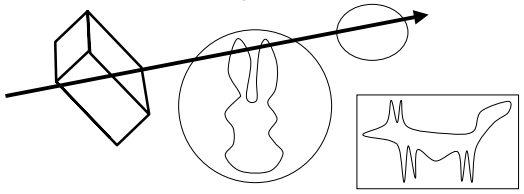
Ray-tracer osztálydiagram



Sugárkövetés: eredmény

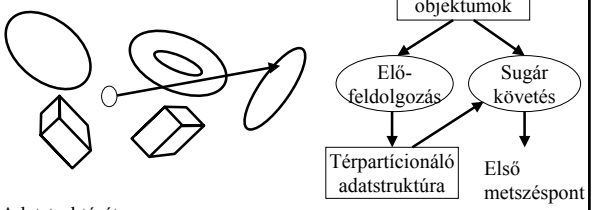
Számítási idő $\propto$ Pixelszám x Objektumszám x (Fényforrás szám+1)

Befoglaló térfogat (Bounding Volume)



```
double IntersectBV( ray, object )      // < 0 ha nincs
    IF ( Intersect( ray, bounding volume of object) < 0) RETURN -1;
    RETURN Intersect( ray, object );
END
```

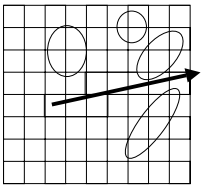
Térpartícionáló módszerek



Adatstruktúrát:

- Ha ismert a sugár, akkor a potenciális metszett objektumok számát csökkenti
- Ha a potenciálisak közül találunk egyet, akkor a többi nem lehet közelebb

Reguláris térháló



Előfeldolgozás:

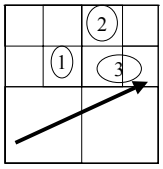
Minden cellára a metszett objektumok komplexitás: $O(n \cdot c) = O(n^2)$

Sugárkövetés:

```
FOR each cell of the line  // line drawing
    Metszés a cellában lévőkkel
    IF van metszés RETURN
ENDFOR
```

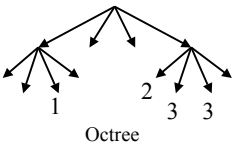
átlagos eset komplexitás: $O(1)$

Nyolcas (oktális) fa



Faépítés(cella):

```
IF a cellában kevés objektum van
    cellában regisztráld az objektumokat
ELSE
    cellafelezés: c1, c2, …, c8
    Faépítés(c1); … Faépítés(c8);
ENDIF
```

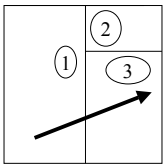


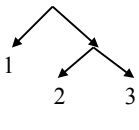
Octree

Sugárkövetés:

```
FOR each cell intersecting the line
    Metszés a cellában lévőkkel
    IF van metszés RETURN
ENDFOR
```

Binary Space Partitioning fa





BSP fa

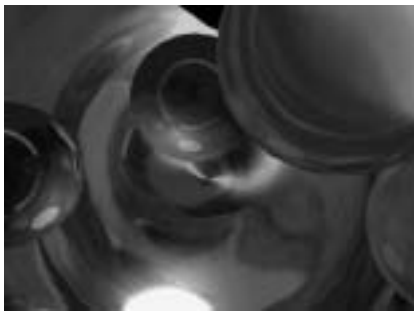
Faépítés(cella):

```
IF a cellában kevés objektum van
    cellában regisztráld az objektumokat
ELSE
    sík keresés
    cella felezés a síkkal: c1, c2
    Faépítés(c1); Faépítés(c2);
ENDIF
```

Sugárkövetés:

```
FOR each cell intersecting the line
    Metszés a cellában lévőkkel
    IF van metszés RETURN
ENDFOR
```

Példa animáció



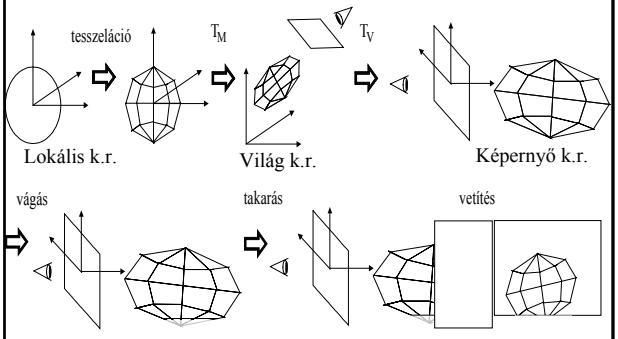
Inkrementális 3D képszintézis

Szirmay-Kalos László

Inkrementális képszintézis

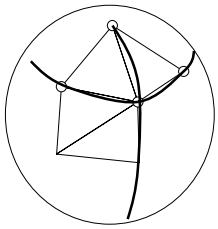
- Árnyalás, láthatóság nehéz, különösen általános helyzetű objektumokra
- koherencia: oldjuk meg nagyobb egységekre
- feleslegesen ne számoljunk: vágás
- transzformációk: minden feladathoz megfelelő koordinátarendszert
  - vágni, transzformálni nem lehet akármit: tesszelláció

3D inkrementális képszintézis



Tesszelláció

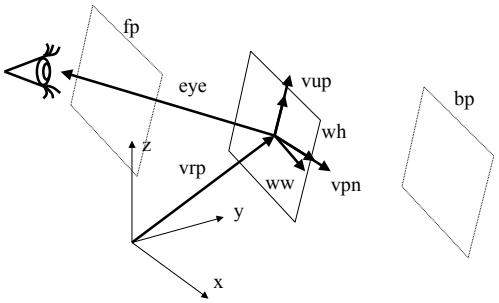
- felületi pontok kijelölése: $r_{n,m} = r(u_n, v_m)$
- felületi pontok összekötése: háromszögháló, huzalváz



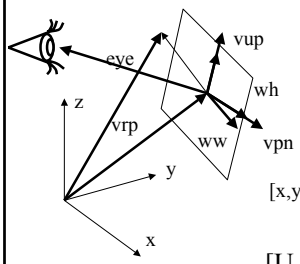
Transzformációk

- Modellezési transzformáció:
  $[r_{lokális}, 1]\ T_M = [r_{világ}, 1]$
- Nézeti transzformáció:
  $[r_{világ}, 1]\ T_v = [r_{képernyő}, 1]$
- Összetett transzformáció:
  $[r_{lokális}, 1]\ T_M T_v = [r_{lokális}, 1]\ T_C = [r_{képernyő}, 1]$

Nézeti transzformáció: Kamera modell



Áttérés az uvw ablak-koordináta rendszerre



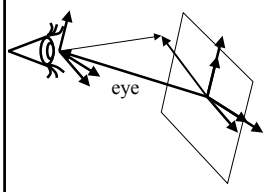
w = vpn/|vpn|
u = w x vup/ |w x vup|
v = u x w

T_{uvw} $\begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ vrp & & & 1 \end{pmatrix}$

$[x,y,z,1] = [U,V,W,1]$

$[U,V,W,1] = [x,y,z,1]\ T_{uvw}^{-1}$

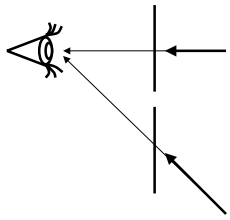
Eltolás a szembe



$r_{eye} = r_{uvw} - eye$

$T_{eye} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -eye_u & -eye_v & -eye_w & 1 \end{pmatrix}$

Nyírás

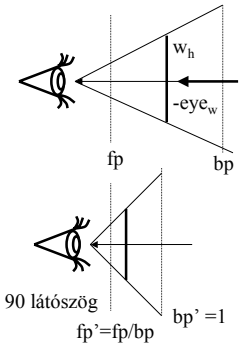


T_{shear} $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ s_u & s_v & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$[eye_u, eye_v, eye_w, 1]\ [0, 0, eye_w, 1]$

$s_u = -eye_u/eye_w$
$s_v = -eye_v/eye_w$

Normalizálás



T_{norm} $\begin{pmatrix} -2eye_w/(w_w\ bp) & 0 & 0 & 0 \\ 0 & -2eye_w/(w_h\ bp) & 0 & 0 \\ 0 & 0 & 1/bp & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

anszformáció

$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & t33 & t34 \\ 0 & 0 & t43 & t44 \end{pmatrix}$

a = fp'
fp'·t33+t43 = 0
fp' ·t34+t44 = 1

T_{persp} $\begin{pmatrix} t11 & t12 & t13 & t14 \\ t21 & t22 & t23 & t24 \\ t31 & t32 & t33 & t34 \\ t41 & t42 & t43 & t44 \end{pmatrix}$

fp'

3 = 1

90 látószög
bp' =1
fp'=fp/bp

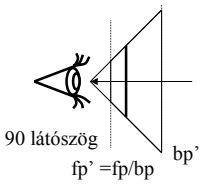
[x, y, 1, 1] * [x y 1 1]
[x, y, fp', 1]* [x/fp', y/fp', 0 1]*a
[0, 0, 0, 1]* [0 0 -1 0]*b

fp'*t33+t43 = 0

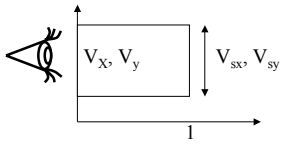
1, 1

$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/(1-fp') & 1 \\ 0 & 0 & -fp'/(1-fp') & 0 \end{pmatrix}$

Perspektív transzformáció



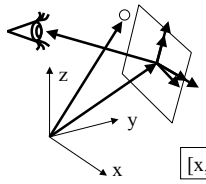
$$T_{persp} = \begin{pmatrix} V_{sx}/2 & 0 & 0 & 0 \\ 0 & V_{sy}/2 & 0 & 0 \\ V_x & V_y & bp/(bp-fp) & 1 \\ 0 & 0 & -fp/(bp-fp) & 0 \end{pmatrix}$$



$$[X_h,Y_h,Z_h,h] = [xc,yc,zc,1]\ T_{persp}$$

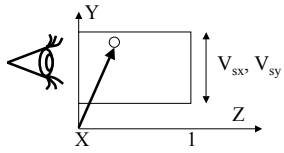
$$[X,Y,Z,1] = [X_h/h,\ Y_h/h,\ Z_h/h,1]$$

Nézeti transzformáció



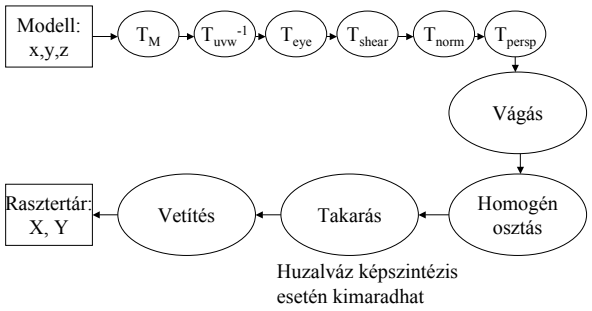
T_V

$$[x,y,z,1]\ T_{uvw}^{-1}\ T_{eye}\ T_{shear}\ T_{norm}\ T_{persp} = [X_h,Y_h,Z_h,h]$$



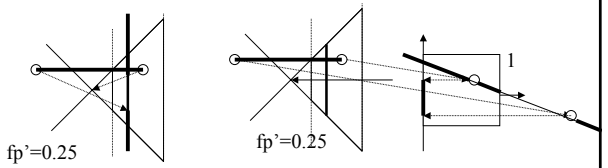
$$[X,Y,Z,1] = [X_h/h,\ Y_h/h,\ Z_h/h,1]$$

Nézeti csővezeték



Huzalváz képszintézis esetén kimaradhat

Mélységi vágás a homogén osztás előtt kell



$$\begin{matrix} 1/(1-fp') \\ \\ -fp'/(1-fp') \\ \\ \end{matrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 4/3 & 1 \\ 0 & 0 & -1/3 & 0 \end{pmatrix}$$

Homogén osztás:
$[X,Y,Z,1] = [X_h/h,\ Y_h/h,\ Z_h/h,1]$

$$[1/4,\ 1/4,\ 1/2,\ 1]^* \begin{bmatrix} 1/4 & 1/4 & 1/3 & 1/2 \end{bmatrix} \quad [1/2\ \ 1/2\ \ 2/3\ \ 1]$$

$$[1/4,\ 1/4,\ -1/2,\ 1]^* \begin{bmatrix} 1/4 & 1/4 & -1 & -1/2 \end{bmatrix} \quad [-1/2\ \ -1/2\ \ 2\ \ 1]$$

Vágás homogén koordinátákban

Cél: $Xmin = V_x - V_{sx}/2 < X = X_h/h < V_x - V_{sx}/2 = Xmax$
$Ymin = V_y - V_{sy}/2 < Y = Y_h/h < V_y - V_{sy}/2 = Ymax$
$Zmin = 0 < Z = Z_h/h < 1 = Zmax$
Vegyük hozzá: $h > 0$ ($h = z$ a kanonikus koordinatarendszerben)

$$Xmin \cdot h < X_h < Xmax \cdot h$$
$$Ymin \cdot h < Y_h < Ymax \cdot h$$
$$Zmin \cdot h < Z_h < Zmax \cdot h$$

$Xmin \cdot h = X_h$
$Xmin = 1$

Belül ○
[3, 0, 0, 2]
$X_{min} \cdot h = 2$
$< X_h = 3$

Kívül ○
[2, 0, 0, 3]
$X_{min} \cdot h = 3$
$> X_h = 2$

Szakasz/poligon vágás

$$Xmin \cdot h < X_h < Xmax \cdot h$$
$$Ymin \cdot h < Y_h < Ymax \cdot h$$
$$Zmin \cdot h < Z_h < Zmax \cdot h$$

$Xmin \cdot h = X_h$ $[X_h^1, Y_h^1, Z_h^1, h^1]$

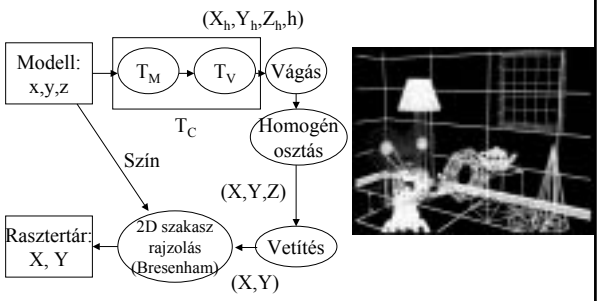
$[X_h^2, Y_h^2, Z_h^2, h^2]$

$$X_{min} \cdot h = X_{min} \cdot (h^1 \cdot (1-t) + h^2 \cdot t) =$$
$$= X_h = X_h^1 \cdot (1-t) + X_h^2 \cdot t$$

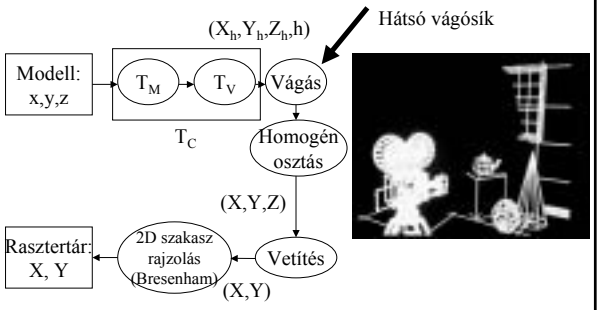
t = …

$$X_h = X_h^1 \cdot (1-t) + X_h^2 \cdot t$$
$$Y_h = Y_h^1 \cdot (1-t) + Y_h^2 \cdot t$$
$$Z_h = Z_h^1 \cdot (1-t) + Z_h^2 \cdot t$$
$$h = h^1 \cdot (1-t) + h^2 \cdot t$$

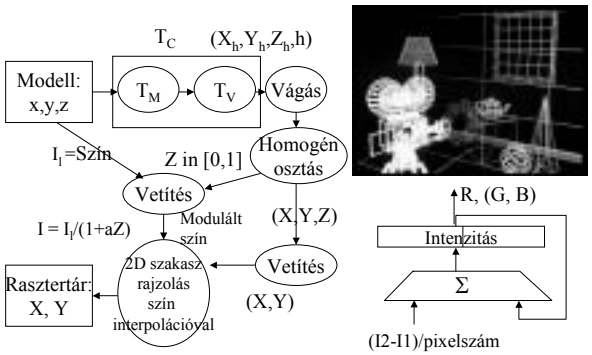
Huzalváz képszintézis



Mélységi vágás



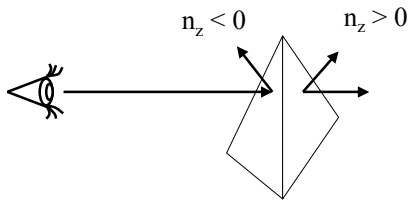
Mélységi intenzitásmoduláció



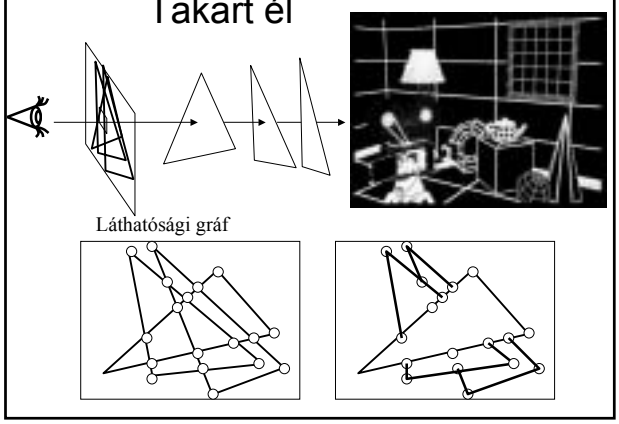
Tömör képszintézés

- Képernyő koordinátarendszerben, a szem a z irányba néz
- Objektumtér algoritmusok: láthatóság számítás nem függ a felbontástól
- Képtér algoritmusok: mi látszik egy pixelben

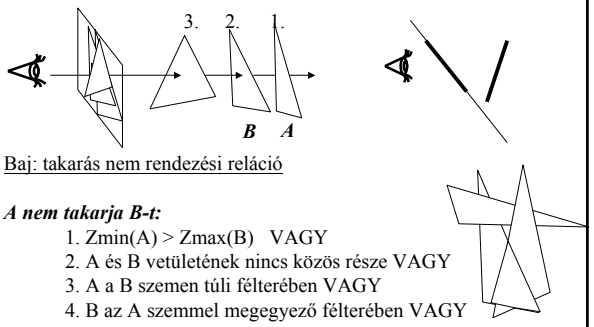
Hátsólab eldobás: back-face culling



Takart él



Festő algoritmus

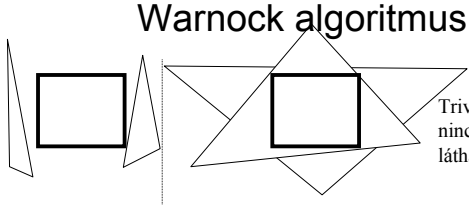


Baj: takarás nem rendezési reláció

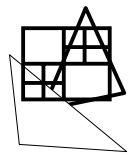
A nem takarja B-t:

1. Zmin(A) > Zmax(B) VAGY
2. A és B vetületének nincs közös része VAGY
3. A a B szemen túli félterében VAGY
4. B az A szemmel megegyező félterében VAGY

Warnock algoritmus

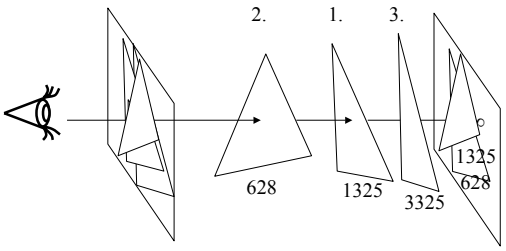


Triviális esetek:
nincs él az ablakban
láthatóság homogén

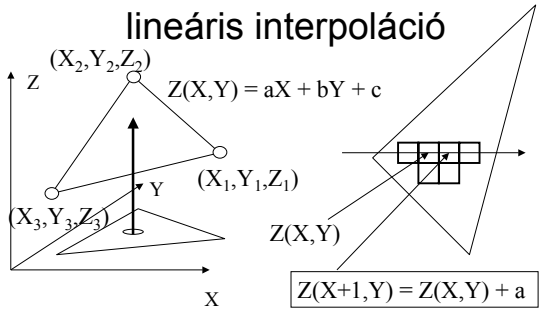


```
Warnock(X1,Y1,X2,Y2) {
    if van él az ablakban && ablak pixelnél nagyobb {
        Xm =(X1+X2)/2; Ym=(Y1+Y2)/2
        Warnock(X1,Y1,Xm,Ym); Warnock(Xm,Y1,X2,Ym);
        Warnock(X1,Ym,Xm,Y2); Warnock(Xm,Ym,X2,Y2);
    } else {
        P = ablak közepén látható poligon
        Ablak kitöltése P színével
    }
}
```

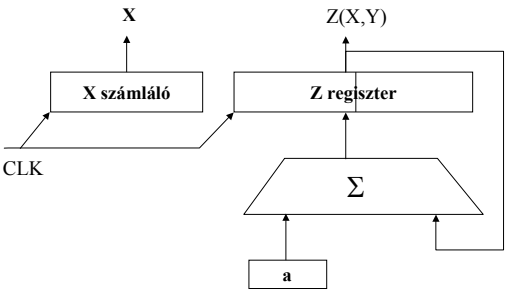
Z-buffer algoritmus



Z-koordináta: lineáris interpoláció



Z-interpolációs hardver



Árnyalás

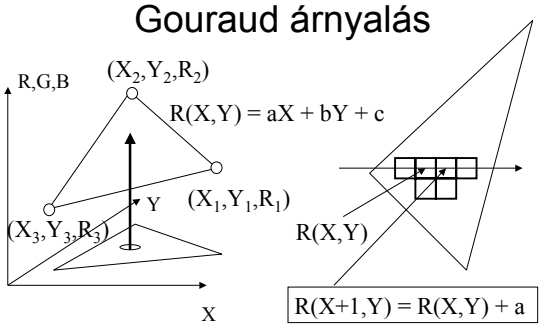
- Koherencia: ne pixelenként kelljen árnyalási egyenletet
- Háromszögenként:
  - 1-szer sem: saját színnel árnyalás
  - 1-szer: konstans árnyalás
  - csúcspontonként 1-szer, belül lineáris interpoláció: Gouraud árnyalás
  - pixelenként, de legalább a normál (view, light, reflection) vektort interpoláljuk: Phong árnyalás

Tömör képszintézis

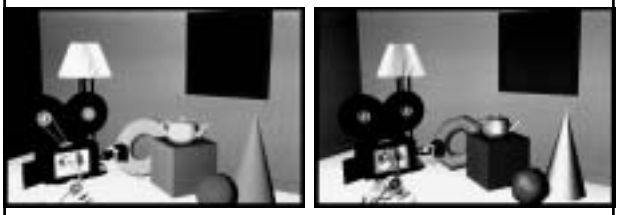
árnyalás saját színnel

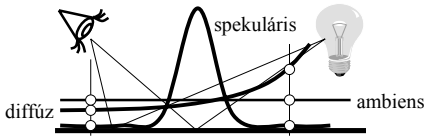
konstans árnyalás

Gouraud árnyalás

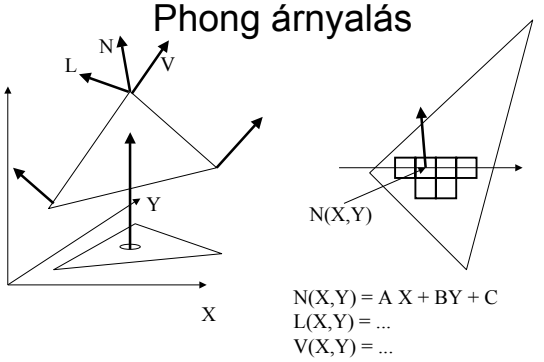


Gouraud árnyalás



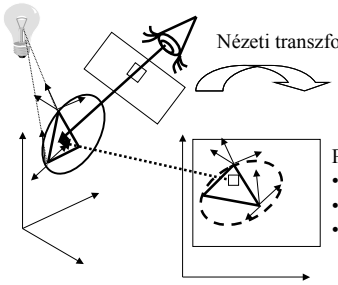


Phong árnyalás



N(X,Y) = A X + BY + C
L(X,Y) = ...
V(X,Y) = ...

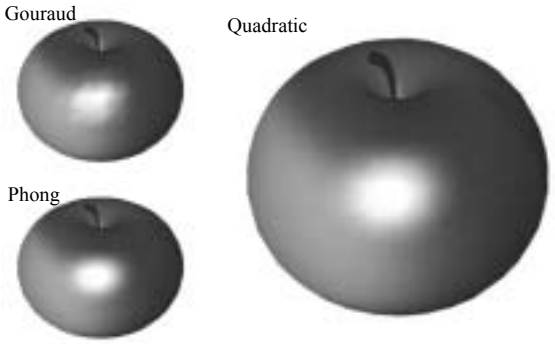
Illumináció: világkoordinátarendszerben



Pixelenként:
- vektorok lin. interpolációja
- vektorok normalizálása
- illuminációs képlet

Phong árnyalás

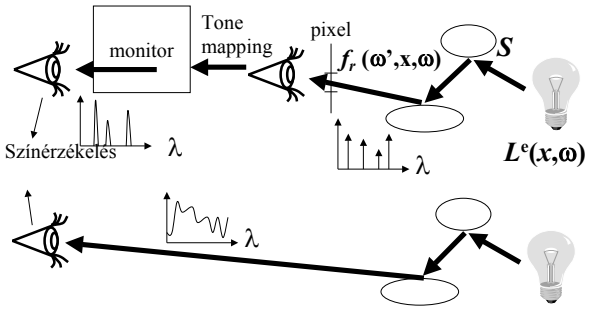
Gouraud, Phong, Quadratic



Globális Illumináció

Szirmay-Kalos László

Képszintézis



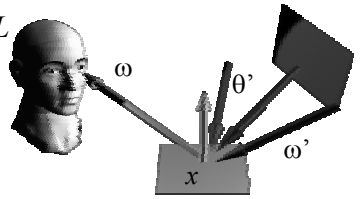
Képszintézis = Árnyalási egyenlet megoldása

Radiancia = emisszió + szóródás (visszaverődés, törés)

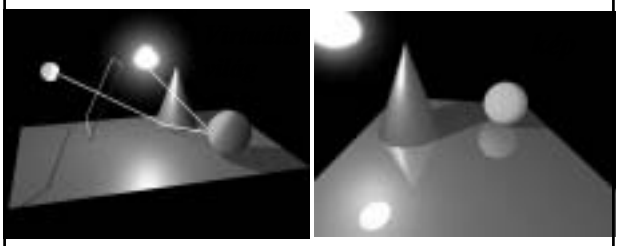
$$L(x,\ \omega) = L^e(x,\ \omega) + \int_\Omega L^{in}(x,\ \omega')\ f_r(\omega',\omega)\ \cos\theta' d\omega'$$

$L = L^e + TL$

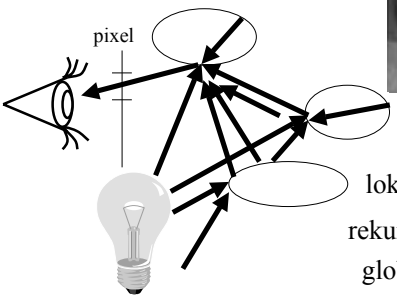
integrál egyenlet



Képszintézis = Integrál a fényutak terében



Követett fényút típusok



lokális illumináció

rekurzív sugárkövetés

globális illumináció

Hogyan működik a természet?

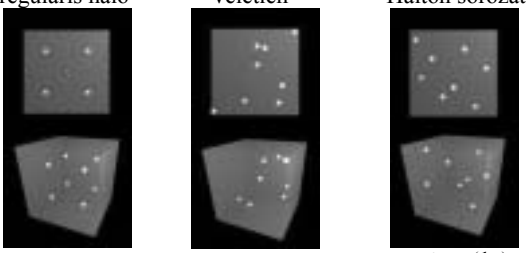
- 100 W égő 10^{42} fotont emittál másodpercenként
- Fotonok által eltalált felület meghatározása fénysebességgel és párhuzamosan
- Fotonok véletlenszerűen elnyelődnek, vagy véletlen irányban visszaverődnek
- A fotonok egy kis része a szembe jut

Szimuláció “néhány” fotonnal

- sokkal kisebb számítási kapacitás
- elfogadható idő: 10^7 foton
- Sűrű minták:
  - fényutak az összes lehetséges régiót lefedik
- Fontosság szerinti mintavételezés:
  - a fényes régiókba több minta
- Kevés minta
  - koherencia

Sűrű minták: $\int f(z)\,dz \approx 1/M \sum f(z_i)$

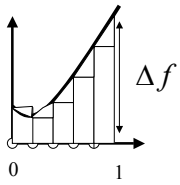
reguláris háló | véletlen | Halton sorozat



Hiba: $\Delta f\ M^{-1/D}$ | $Df\ M^{-1/2}$ | $\Delta f\ M^{-(1-\varepsilon)}$

Klasszikus num. integrálás | Monte-Carlo integrálás | Kvázi-Monte-Carlo integrálás

Téglány szabály



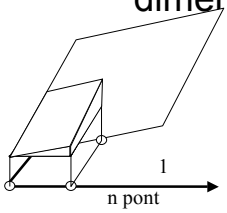
$\int f(z)\,dz \approx 1/M \sum f(z_i)$

M db minta

Átlagos magasság · alap · Háromszögek száma

Hiba = $\Delta f/2/M \cdot 1/M \cdot M = \Delta f/2/M = O(M^{-1})$

Téglányszabály magasabb dimenziókban



$M=n^2$ minta

$n = \sqrt{M}$

Átlagos magasság · alap · Tetők száma

Error = $\Delta f/2/n \cdot 1/n^2 \cdot n^2 = \Delta f/2/n = O(M^{-0.5})$

Klasszikus numerikus integrálás D dimenzióban

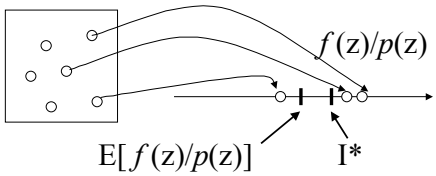
- Hiba: $\Delta f/2\ M^{-1/D} = O(M^{-1/D})$
- Szükséges mintaszám:

  $O(\ (\Delta f/error)^D\ )$
- **Exponenciális robbanás!**

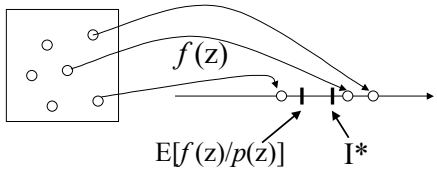
Monte-Carlo integrál

Vezessük vissza az integrálást várható érték számításra

$$I = \int f(z)\,dz = \int f(z)/p(z)\cdot p(z)\,dz = E[f(z)/p(z)] \approx 1/M\,\Sigma f(z_m)/p(z_m) = I^*$$



Várhatóérték becslés átlaggal

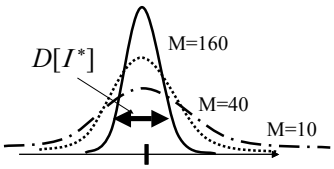


$D^2[I^*] = D^2[1/M \cdot \Sigma f(z_m)/p(z_m)] = 1/M^2 \cdot D^2[\Sigma f(z_m)/p(z_m)] =$

Ha a minták függetlenek! :

$1/M^2 \cdot M \cdot D^2[f(z)/p(z)] = 1/M \cdot D^2[f(z)/p(z)]$

Az I^* átlag eloszlása



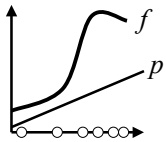
$\Pr\{ |I^* - E[I^*]| < 3D[I^*] \} > 0.997$

Statisztikai hibakorlát (99.7% konfidenciával):

$$|I^* - I| < 3D[f/p]/\sqrt{M}$$

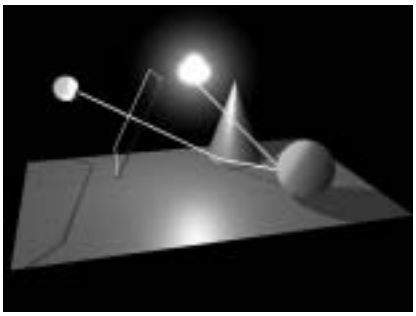
Monte-Carlo integrálás

- A hiba: $3D[f/p]/\sqrt{M}$ a dimenziótól függetlenül!
- Az f/p szórásának kicsinek kell lennie
  - ahol f nagy, ott p is legyen nagy

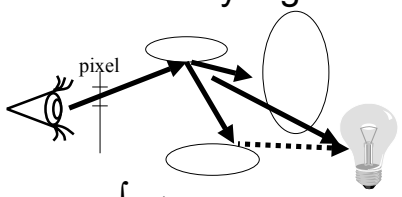


Fontosság szerinti mintavételezés

Fényutak mélységi keresése: Bolyongás: $L = L^e + TL = \Sigma T^n L^e$



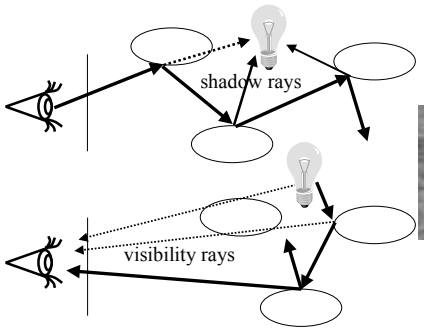
Fontosság szerinti mintavételezés a véletlen bolyongásban



$$L = L^e + \int_\Omega L^{in}(\omega')\, f_r \cos\theta' d\omega'$$

- BRDF mintavételezés: $f_r \cos\theta'$-val arányosan
- Fényforrás mintavételezés

A fényutak felépítési iránya



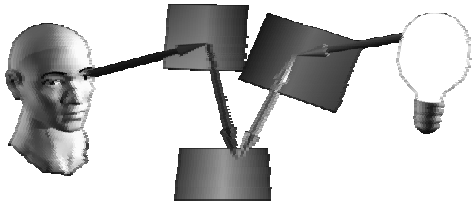
Nincs "caustics"

gathering

shooting

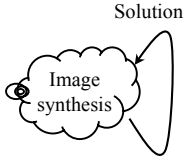
Nem jó látható üvegre és tükörre

Bi-directional Path Tracing



Véletlen bolyongás analízise

- **Előnyök**
  - eredeti geometria
  - nincs hiba akkumuláció
  - triviális párhuzamosítás
- **Hátrányok**
  - nem struktúrált geometriai lekérdezések
  - nézőpont függő
  - $O(M^{-0.5})$ konvergencia

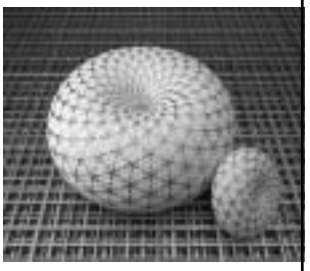


Lassú mert nem használja ki a koherenciát

Véletlen bolyongás példák

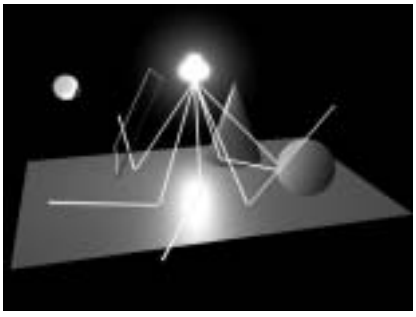
7 felületelem

100 minta / 23 min



sok óra (Kelemen Csaba)

Szélesség szerinti keresés Iteráció: $L_n = L^e + T L_{n-1}$



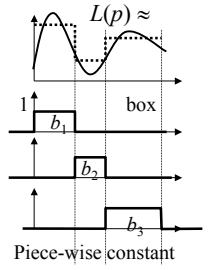
A sugársűrűség ideiglenes tárolása: Véges-elem módszer

- Sugársűrűség véges elem közelítése:

  $\Sigma L_j b_j(p)$

- Integrálegyenletből egy lin. egyenletrendszer lesz

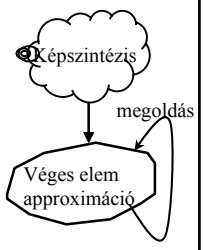
$$\mathbf{L}^{(n)} = \mathbf{L}^e + \mathbf{R}\,\mathbf{L}^{(n-1)}$$



Iteráció analízise

- **Előnyök**
  - koherencia kihasználása
  - hardver láthatóság
  - nézet függő
  - $O(a^M)$ konvergencia
- **Hátrányok**
  - Véges-elem közelítése
  - nem triviál párhuzamosítás
  - hiba akkumuláció
  - nagy tárolóigény

Lassú a sok véges elem miatt



Sztochasztikus iteráció

- Véletlen operátor T^*
  - $L_n = L^e + T_n * L_{n-1}$
- átlagban visszaadja az eredeti operátor hatását
  - $E[T_n * L] = T\ L$
- $C_{pixel} = M\ L_n$ pixel színek nem konvergensek
  $C_{pixel}(m) = (M\ L_1 + M\ L_2 + ... + M\ L_m)/m$

Példa: x = 0.1 x + 1.8

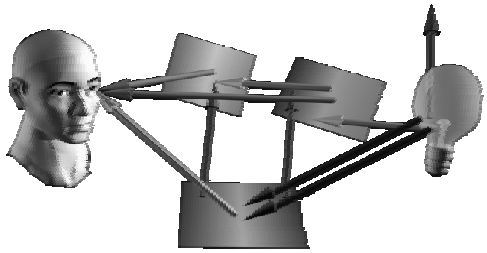
- Véletlen operátor:
  $x_n = T_n\ x_n + 1.8, \quad T$ v.v. $[0, 0.2]$

n:	1	2	3	4	5
T_n sorozat:	0	0.1	0.2	0.15	0.05
x_n sorozat: 0 $\Rightarrow$	1.8	1.91	2.18	2.13	1.9
Átlagolás:	1.8	1.85	1.9	2.04	1.96

x_n sorozat: nem konvergens!

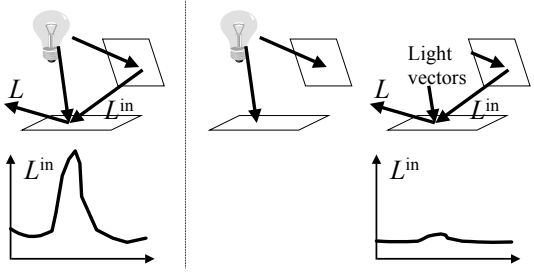
Sztochasztikus iteráció sugárkötegekkel

T*: *Összes pont sugársűrűségének átvitele egy véletlen irányban*



Első-lövés

Integrálási hiba $< 3D[\,f/p\,] / \sqrt{M}$



Szimulációs eredmények

15 K felület
PIII 500 Mhz

First-shot + 40 iterations

51 secs

Fraktális magasságmező: 60 K

Első lövés: 30 sec

3 min

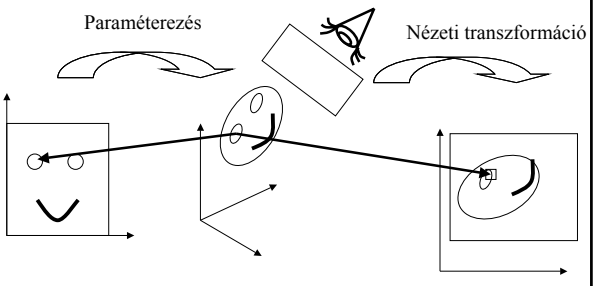
ArchiCad (Graphisoft) model

Textúra leképzés

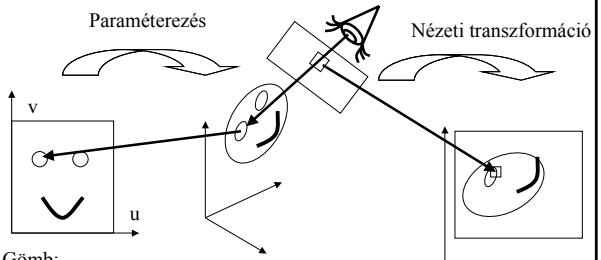
Szirmay-Kalos László

Textúra leképzés:
anyagjellemzők változnak a felületen

Textúra leképzés



Textúra leképzés sugárkövetésnél

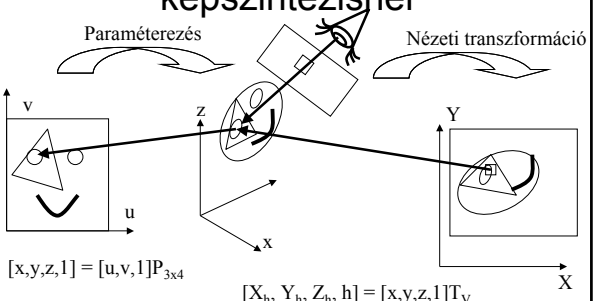


Gömb:

$x = x0 + r \cos 2\pi u \sin \pi v$
$y = y0 + r \sin 2\pi u \sin \pi v$
$z = z0 + r \cos \pi v \qquad u,v \in [0,1]$

$u = 1/2\pi \ \ \text{atan2}((x-x0), (y-y0))$
$v = 1/\pi \ \ \text{acos}((z-z0)/r)$

Textúra leképzés inkrementális képszintézisnél



$[x,y,z,1] = [u,v,1]P_{3x4}$

$[X_h, Y_h, Z_h, h] = [x,y,z,1]T_V$

$[X,Y] = [X_h/h, Y_h/h]$

Textúra mátrix

Textúra térből képernyő koordinátarendszerbe:

$[u,v,1]\ P_{3x4}\ T_V = [X_h, Y_h, Z_h, h]$

Redundáns a mélység érték:

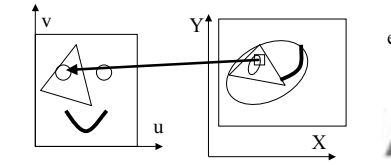
$[u,v,1]\ P_{3x4}\ T_V = [X_h, Y_h, \cancel{Z_h}, h] \Rightarrow [u,v,1]\ \overbrace{(P_{3x4}\ T_V)}^{3x3} = [X_h, Y_h, h]$

Pixel koordinátákra kifejezve:

$[u/h, v/h, 1/h]\ \overbrace{(P_{3x4}\ T_V)}^{3x3} = [X_h/h,\ Y_h/h,\ 1]$

Invertálva: $[U,V,W] = [X,Y,1]\ T$

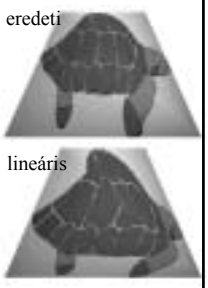
Textúra leképzés inkrementális képszintézisben



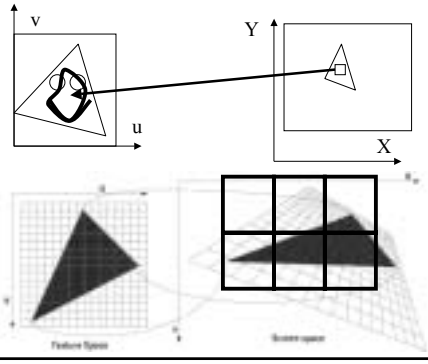
$U = aX + bY + c$
$V = eX + fY + g$
$W = hX + iY + j$

$u = \frac{aX + bY + c}{hX + iY + j}$

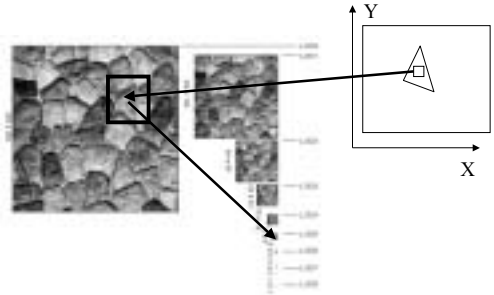
$v = \frac{eX + fY + g}{hX + iY + j}$



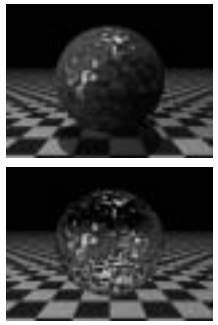
Textúrák szűrése

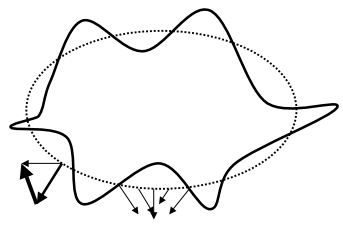


Mip-map adatstruktúra



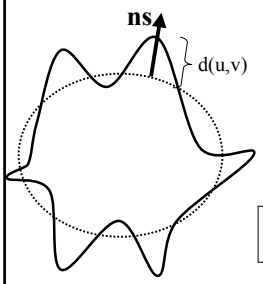
Bucka leképzés (Bump mapping)





Tároljuk a normálvektor perturbációt táblázatban vagy a felületi elmozdulást

Normálvektor perturbáció a magasság perturbációból



s(u,v): sima felület
r(u,v): rücskös felület

$\mathbf{r}(u,v) = \mathbf{s}(u,v) + d(u,v)\ \mathbf{ns}$

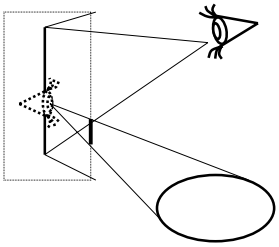
$\mathbf{nr} = \mathbf{r}_u \times \mathbf{r}_v =$
$(\mathbf{s}_u + d_u\mathbf{ns} + d\ \cancel{\mathbf{ns}_u}) \times (\mathbf{s}_v + d_v\mathbf{ns} + d\ \cancel{\mathbf{ns}_v}) =$

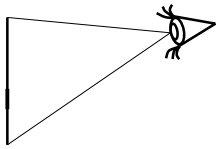
$\mathbf{s}_u \times \mathbf{s}_v + (d_u\mathbf{ns} \times \mathbf{s}_v + d_v\,\mathbf{s}_u \times \mathbf{ns})$

ns Normálvektor perturbáció

Magasság perturbáció pontosan displacement mapping

Környezet leképzés: environment map





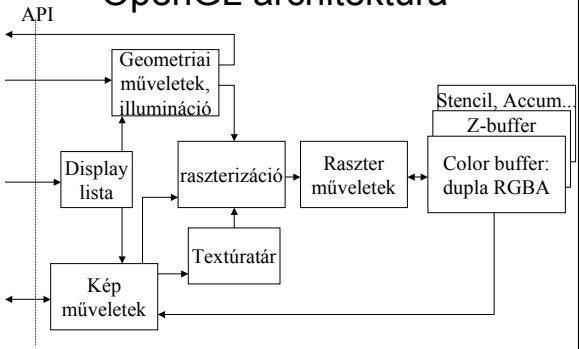
OpenGL

Szirmay-Kalos László

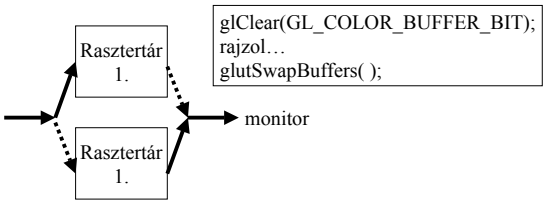
OpenGL

- Képszintézis könyvtár (2D, 3D), API
  - 2D, 3D geometria
  - rajzolási állapot (state)
  - raszterműveletek
- Op. Rendszer, Ablakozó rendszer független
  - ő nem ablakoz

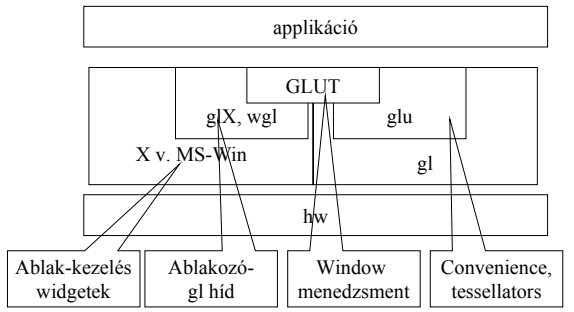
OpenGL architektúra



Dupla buffer animációhoz



OpenGL elhelyezkedése



OpenGL szintaxis

gl könyvtár része

```
glVertex3dv( … )
```

Paraméterszám	***Adattípus***	***Vektor vagy skalár***
2 - (x, y)	b - byte	v - vektor
3 - (x, y, z), (R, G, B)	ub - unsigned byte	- skalár
4 - (x, y, z, h) (R, G, B, A)	s - short	
	i - int	
	f - float	
	d - double	

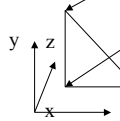
OpenGL alkalmazás (GLUT)

- Ablak megnyitás
- Rajzolási állapot, display lista inicializálás
- Display callback:
  - képernyő törlés, állapotváltás, rajzolás, buffercsere
- Reshape callback:
  - nézeti transzformácó, vágás átállítása
- Input callback (mouse, keyboard)
- Idle callback (animáció)
- Üzenethurok

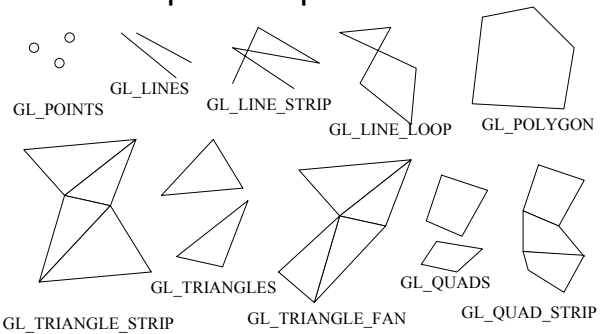
OpenGL: inicializálás

```
glutInitWindowSize(200, 200);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_RGBA |
   GLUT_DOUBLE | GLUT_DEPTH);
glutCreateWindow("Sample Window");
  // állapot, transzformáció inicializálás
glShadeModel(GL_SMOOTH);
glEnable(GL_DEPTH_TEST);
glEnable(GL_DITHER);
                // callback függvények
glutKeyboardFunc( Keyboard );
glutDisplayFunc( ReDraw );
glutMainLoop();     // fő hurok
```

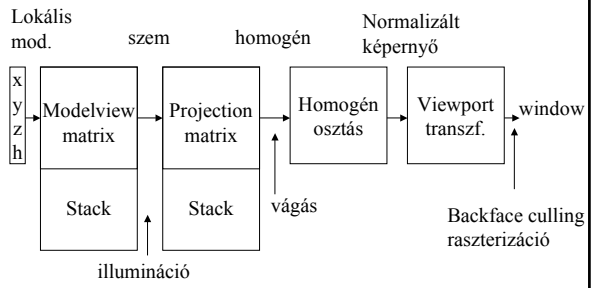
```
void ReDraw( ) {
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT|
          GL_DEPTH_BUFFER_BIT);
  glBegin(GL_TRIANGLES);
  glColor3d(0.0, 0.0, 1.0);
  glVertex3d(10.0, 20.0, 10.0);
  glColor3d(1.0, 0.0, 0.0);
  glVertex3d(10.0, 10.0, 10.0);
  glVertex3d(20.0, 10.0, 10.0);
  glEnd( );
  glFlush( );
  glutSwapBuffers( );
}
```



OpenGL: primitívek



Transzformációk



Transzformációk beállítása

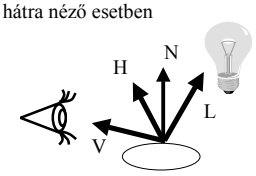
```
void resize(int w, int h) {
   glViewport(0, 0, w, h);                    // viewport transzformáció
   glMatrixMode( GL_PROJECTION);  // projection transzformáció
   glLoadIdentity( );
   gluPerspective( fov, (double)w/h, front, back);
   glMatrixMode( GL_MODELVIEW);  // modelview transzformáció
   glLoadIdentity( );
                                              // modelview/view
   gluLookAt(eyex, eyey, eyez, vrpx, vrpy, vrpz, vupx, vupy, vupz);
   glTranslated(0, 0, -10);                   // modelview/model
   glRotated(angle, axisx, axisy, axisy);
}
```

Árnyalás

Phong-Blinn model

Color = Emission +
Ambient · Ia +
Diffuse ·(N ·L) ·Id +
Specular ·(N ·H)^{shininess} ·Is

Különböző paraméterek az előre és hátra néző esetben




```
Glfloat kd[] = {0.0, 1.0, 1.0, 1.0};
Glfloat ks[] = {1.0, 0.5, 1.0, 1.0};
Glfloat ka[] = {0.2, 0.0, 0.2, 1.0};

glMaterialfv( GL_FRONT,
              GL_AMBIENT, ka);
glMaterialfv( GL_FRONT,
              GL_DIFFUSE, kd);
glMaterialfv( GL_FRONT,
              GL_SPECULAR, ks);
glMaterialf( GL_FRONT,
             GL_SHININESS, 20);
```

```
glEnable( GL_NORMALIZE);

glBegin(GL_TRIANGLES);

glNormal3d(0.0, 0.0, 1.0);
glVertex3d(10.0, 20.0, 10.0);

glNormal3d(1.0, 0.0, 0.0);
glVertex3d(10.0, 10.0, 10.0);
….
```

Fényforrások

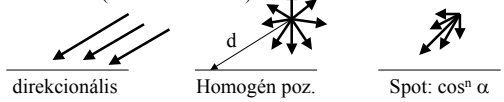
```
Glfloat I[] = {0.0, 1.0, 1.0, 0};
Glfloat pos[] = {10, 20, 10, 1.0 };

glLightfv(GL_LIGHT0, GL_DIFFUSE, I);
glLightfv(GL_LIGHT1,….

glEnable(GL_LIGHT0 );
glEnable(GL_LIGHTING );
```

Ha h = 0, akkor direkcionális egyébként pozicionális

Koordinátarendszer:

Modelview: fényszóró
View: világ
Külön Modelview: animált

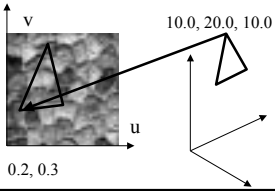
- Külön fényforrás az ambiens, diffúz, spekuláris visszaverődéshez
- Direkcionális, Pozicionális (homogén ill. spot)
- Attenuation: $1/(k0+ k1 \cdot d + k2 \cdot d^2)$



Textúrák

- Textúrák definiálása (texture object)
- Opcionális textúrakiegészítők:
  - Textúra szűrő, paraméterező transzf, wrap, perspektív korrekció
- Textúra kiválasztása
- Textúrázás engedélyezése
- Csúcsokhoz textúrakoordináták megadása vagy számíttatása

```
glBegin(GL_TRIANGLES);
glTexCoord2d(0.2, 0.3);
glVertex3d(10.0, 20.0, 10.0);
glTexCoord2d(0.5, 0.8);
glVertex3d(10.0, 10.0, 10.0);
```

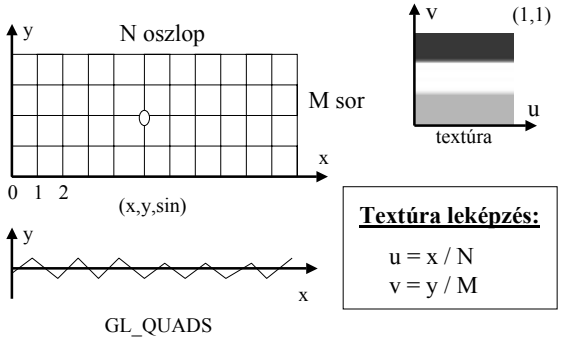


Textúra objektum létrehozása

```
int texids[10];
glGenTextures(10, texids);

glBindTexture( GL_TEXTURE_2D, textids[0] );
int level = 0, border = 0, width = 256, height = 256; // 2 hatvány
unsigned char image[256*256*3]; // kitöltés

glTextImage2D(GL_TEXTURE_2D, level, GL_RGB, width, height,
              border, GL_RGB, GL_UNSIGNED_BYTE, &image[0]);

      // opcionális paraméterező transzformáció, vagy csúcsonkénti u, v
glEnable(GL_TEXTURE_GEN_S); glEnable(GL_TEXTURE_GEN_T);
float ttu[ ] = {0.1, 0.3, 0.6, 2}, ttv = {0.1, 0.3, 0.6, 2};

glTexGenfv( S, ttu, OBJECT_LINEAR );
```

u = ax+by+cz+d

Példa: zászló



Textúra leképzés:

u = x / N
v = y / M

Inicializálás

```
main( argc, argv ) {
   glutInitWindowSize(200, 200);  glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE |GLUT_DEPTH);
   glutCreateWindow(”Waving flag");
   glViewport(0, 0, width, height);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(54.0,(GLfloat)width/(GLfloat)height,1.0,1000.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslatef(-15.0, -10.0, -50.0);
      Geometria inicializálás, Textúra betöltés
   glutIdleFunc( Render );    // Idle callback regisztráció
   glutMainLoop();
}
```

Geometria inicializálás

```
Vector flag[N+1][M+1];
int    texture;
for (int x = 0; x <= N; x++) {
      for (int y = 0; y <= M; y++) {
            flag[x][y].x = x;
            flag[x][y].y = y;
            flag[x][y].z = sin(x*20.0/180.0*M_PI);
      }
}
```

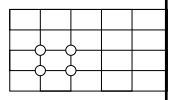
Textúra betöltés

```
glEnable(GL_TEXTURE_2D);

unsigned char bitmap[3*maxwidth*maxheight];
LoadBitmap(”lobogo.bmp", &bitmap, &width, &height);

glGenTextures(1, &texture);                  // azonosító gen.
glBindTexture(GL_TEXTURE_2D, texture);       // kötés

glTexImage2D(GL_TEXTURE_2D,
                  0, GL_RGB, width, height, 0,
                  GL_RGB, GL_UNSIGNED_BYTE, bitmap);
```

Képszintézis, ha éppen ráér

```
void Render() {
      glClear(GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT);
      DrawFlag();
      glFlush();

      glutSwapBuffers( );
      AnimateFlag( );
}
```

DrawFlag



```
DrawFlag( ) {
   glBegin(GL_QUADS);
   for (int x = 0; x < N; x++)  for (y = 0; y < M; y++) {
      glTexCoord2f( x/N, y/M );
      glVertex3f( flag[x][y].x, flag[x][y].y, flag[x][y].z );
      glTexCoord2f( (x+1)/N, y/M );
      glVertex3f( flag[x+1].x, flag[x+1][y].y, flag[x+1][y].z );
      glTexCoord2f( (x+1)/N, (y+1)/M );
      glVertex3f( flag[x+1][y+1].x, flag[x+1][y+1].y, flag[x+1][y+1].z);
      glTexCoord2f( x/N, (y+1)/M );
      glVertex3f( flag[x][y+1].x, flag[x][y+1].y, flag[x][y+1].z );
   }
   glEnd();
}
```

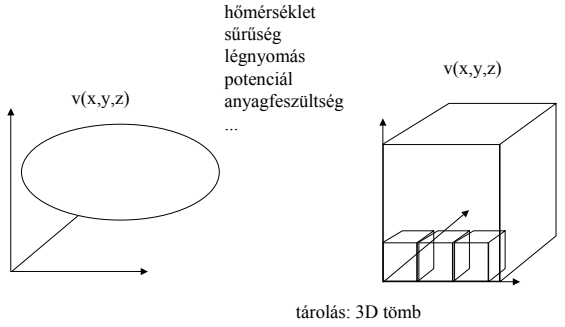
AnimateFlag

```
AnimateFlag( ) {
    static float phase = 0;
    for (int x = 0; x <= N; x++) {
        for (int y = 0; y <= M; y++) {
            flag[x][y].x = x;
            flag[x][y].y = y;
            flag[x][y].z = sin(x*20.0/180.0*M_PI + phase);
        }
    }
    phase += 2 * M_PI / 10;
}
```

Térfogatvizualizáció

Szirmay-Kalos László

Térfogati modellek

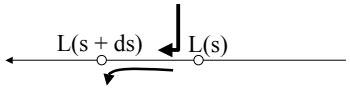


tárolás: 3D tömb

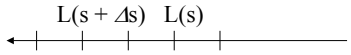
Térfogati modell megjelenítése

- Kocka megjelenítése ???
- Megjelenítés átlátszó anyagként (belsejébe belelátunk)
- Adott szintfelület kiemelése (külsôt lehámozzuk)

Átlátszó anyagok

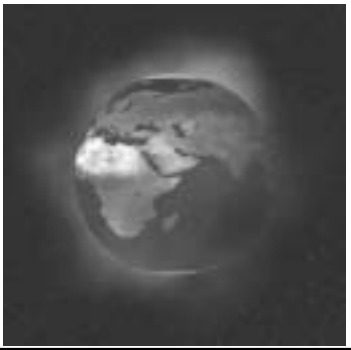


$$dL(s)/ds = - kt \cdot L(s) + ka \cdot L^e + \int f(\omega', \omega)\, L^i(\omega')\, d\,\omega'$$



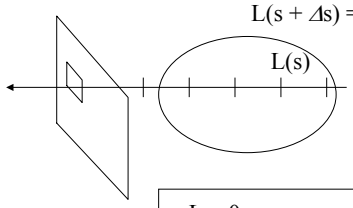
$$L(s + \Delta s) = L(s) - \underbrace{kt\,\Delta s}_{\alpha(s)} \cdot L(s) + \underbrace{Li(s)\,\Delta s}_{C(s)}$$

Fényemittáló, fényelnyelő anyag



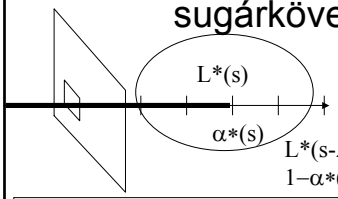
Számítás fénysugárkövetéssel

$L(s + \Delta s) = (1- \alpha(s)) \cdot L(s) + C(s)$




```
L = 0
for(s = 0; s < T; s += Δs) {
        L = (1- α(s)) · L + C(s)
}
```

Számítás láthatóság sugárkövetéssel

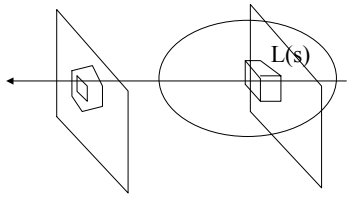


$L^*(s-\Delta s)=L^*(s)+(1- \alpha^*(s)) \cdot C(s)$

$1-\alpha^*(s-\Delta s)=(1-\alpha^*(s)) \cdot (1-\alpha(s))$

```
L* = α* = 0
for( s = T; s >0 ; s -= Δs ) {
        L += (1- α*) · C(s)
        α* = 1–(1- α*) · (1–α(s))
        if (α* > 1–ε) break
}
```

Térfogat vetítés

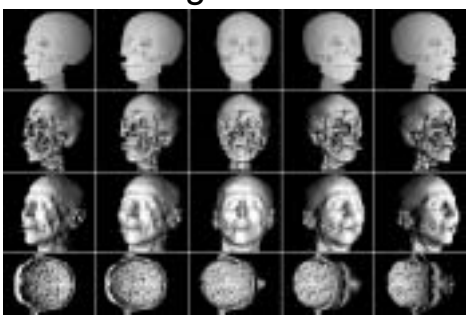


$L(s + \Delta s) = (1- \alpha(s)) \cdot L(s) + C(s)$

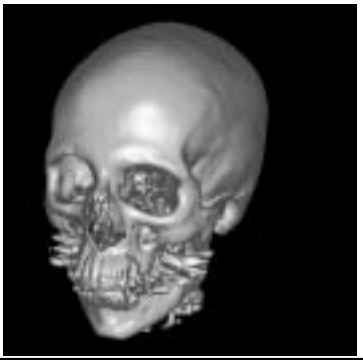
Voxel szín és opacitás

- Röntgen:
  - opacitás = v(x,y,z)
  - C(0) = 1, egyébként 0
- Klasszikus árnyalási modellek
  - opacitás: v osztályozása
  - C = árnyalási modell (diffúz + Phong)
    - normál = grad v
    - opacitás *= | grad v |

Különböző árnyalási modellek, szegmentálás



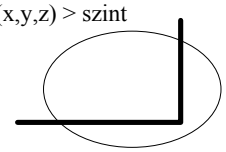
Phong árnyalás



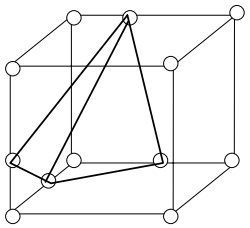
Maximum intenzitás vetítés

Marching cubes

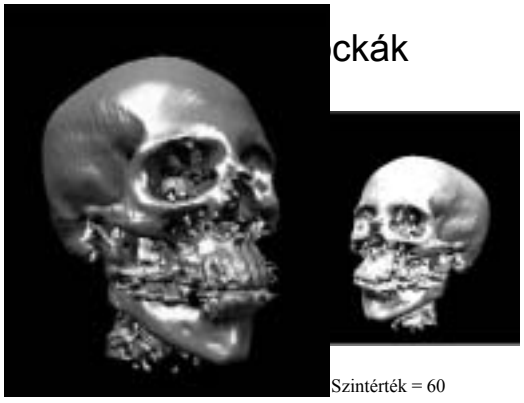
v(x,y,z) > szint



v(x,y,z) < szint

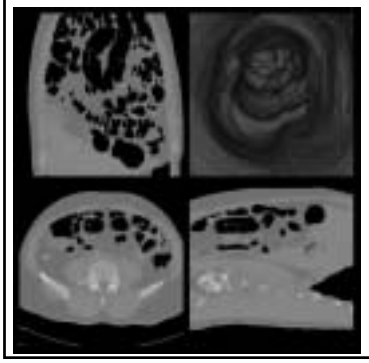


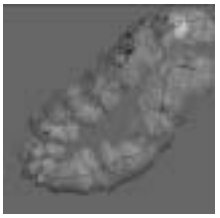
ckák



Szintérték = 60

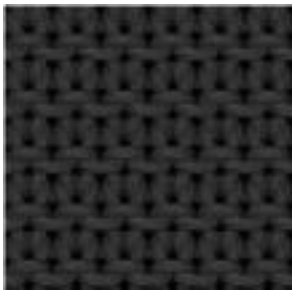
Virtuális endoszkópia



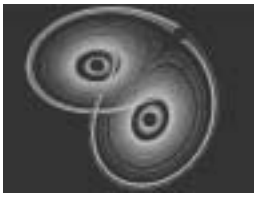


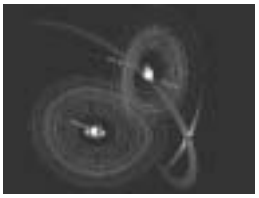
Jocha Dávid
Koloszár József

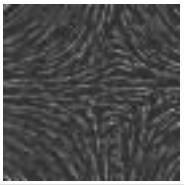
Szövetek megjelenítése



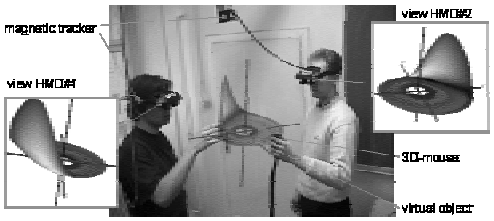
Mérnöki vizualizáció







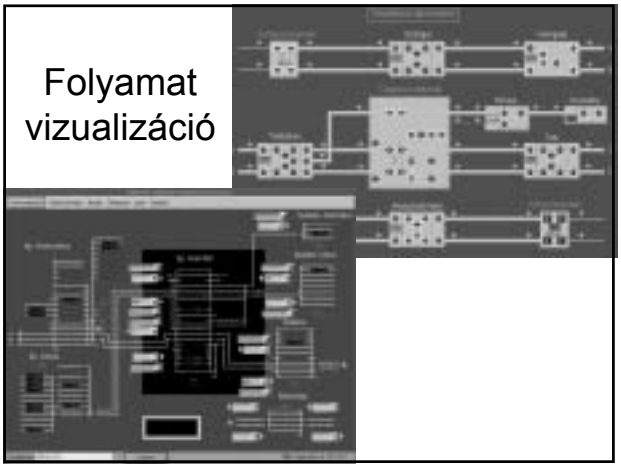
Augmentált valóság



Augmentált valóság

Blaskó Gábor

Folyamat vizualizáció



Fraktálok

Szirmay-Kalos László

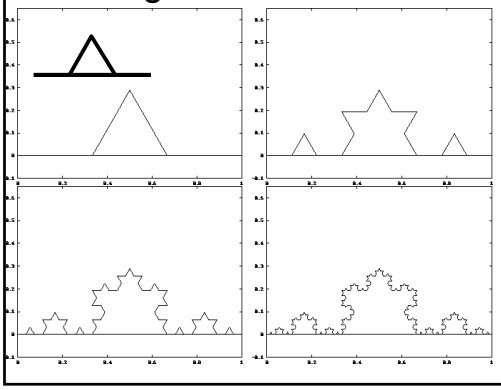
Fraktálok

Hausdorff dimenzió

$N = 1/r^D$ $D = (\log N) / (\log 1/r)$

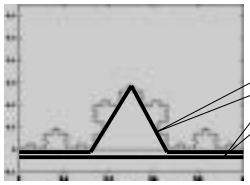
Koch görbe

$D = (\log 4) / (\log 3) = 1.26$



N = 4,
r = 1/3

Nem önhasonló objektumok dimenziója

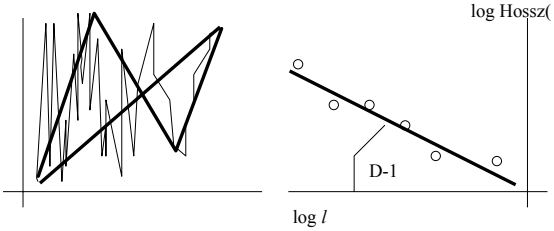


Vonalzó (l)	db
1	1
r = 1/3	N = 4
r^2	N^2
r^m	N^m

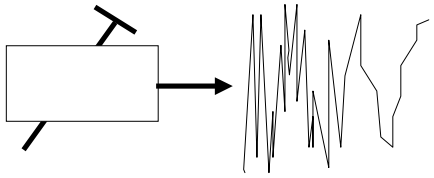
Hossz(l) = l db = l N^m = l $(1/r^D)^m$ =
= l $(1/r^m)^D$ = $1/l^{D-1}$

$$D = - \log \text{Hossz}(l) / \log l + 1$$

Dimenziómérés = hosszmérés



Fraktálok előállítása

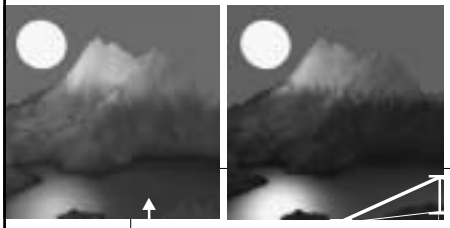


Matematikai gépek:
- Brown mozgás
- Kaotikus dinamikus rendszerek

Brown mozgás - Wiener féle sztochasztikus folyamat

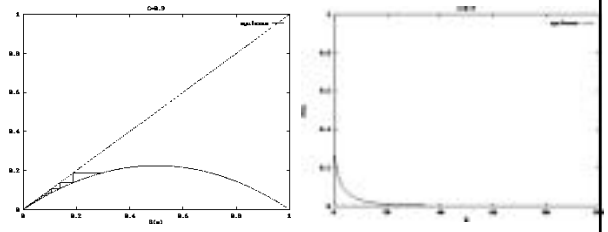
- Sztochasztikus folyamat (véletlen függvény)
- Trajektóriák folytonosak
- Független növekményű folyamat
- Növekmények 0 várható értékű normális eloszlás:
  - a független növekményűségből, a szórás az intervallum hosszával arányos

Brown mozgás alkalmazása

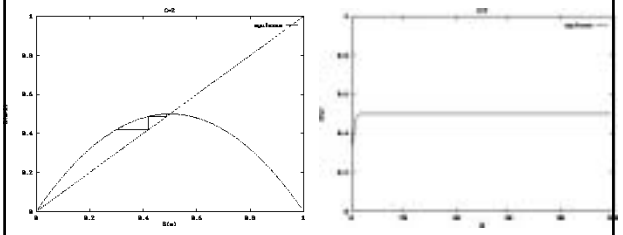


Kaotikus dinamikus rendszer: nyulak kis C értékre

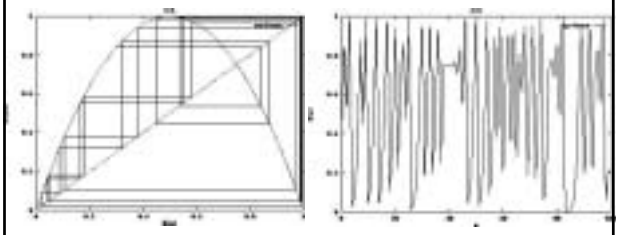
$$S_{n+1} = C\, S_n\, (1-S_n)$$



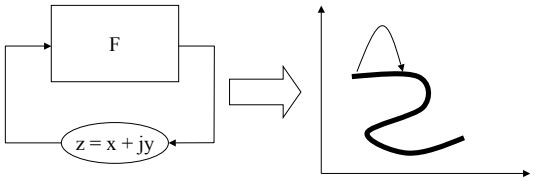
Kaotikus dinamikus rendszer: nyulak közepes C értékre



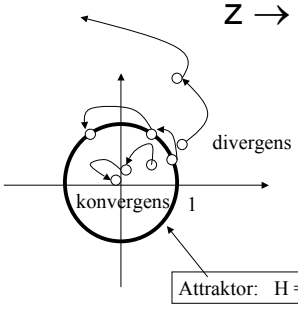
Kaotikus dinamikus rendszer: nyulak nagy C értékre



Kaotikus rendszerek a síkon



$z \rightarrow z^2$



$z = r\, e^{i\phi}$

$r \rightarrow r^2$

$\phi \rightarrow 2\phi$

Attraktor előállítása

- Attraktor a labilis és a stabilis tartomány határa: kitöltött attraktor = amely nem divergens
  - $z_{n+1} = z_n^2$: ha $z_\infty < \infty$ akkor fekete
- Attraktorhoz konvergálunk, ha az stabil
  - $z_{n+1} = z_n^2$ attraktora labilis

Inverz iterációs módszer

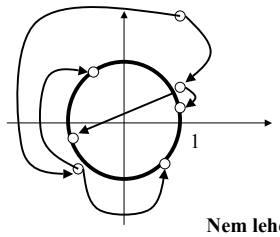
$H = F(H) \rightarrow H = F^{-1}(H)$

$z_{n+1} = z_n^2$

$z_{n+1} = \pm \sqrt{z_n}$

$r_{n+1} = \sqrt{r_n}$

$\phi_{n+1} = \phi_n/2 + \{0,1\}\cdot\pi$



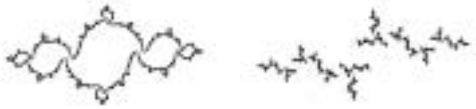
$r_n \approx 1$

$\phi_n \approx \{0,1\}.\{0,1\}\{0,1\}\ldots \cdot\pi$

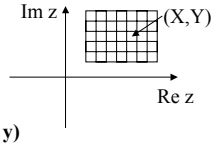
n n-1 n-2

Nem lehet csak egy értékkel dolgozni ???

Julia halmaz: $z \rightarrow z^2 + c$

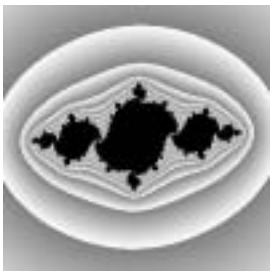


Kitöltött Julia halmaz: algoritmus

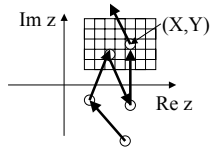


FilledJuliaDraw ( )
FOR Y = 0 TO Ymax DO
FOR X = 0 TO Xmax DO
ViewportWindow(X,Y → x, y)
z = x + j y
FOR i = 0 TO n DO z = z^2 + c
IF |z| > "infinity" THEN WRITE(X,Y, white)
ELSE WRITE(X,Y, black)
ENDFOR
ENDFOR
END

Kitöltött Julia halmaz: kép



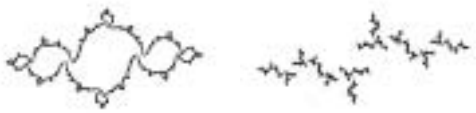
Julia halmaz inverz iterációval



JuliaDrawInverseIterate ( )
Kezdeti z érték választás
FOR i = 0 TO n DO
x = Re z, y = Im z
IF ClipWindow(x, y)
WindowViewport(x, y ⇨ X, Y)
Pixel(X, Y) = fekete
ENDIF
z = $\sqrt{z - c}$
if (rand( ) > 0.5) z = -z
ENDFOR
END

Kezdeti z érték: $z^2 = z - c$ gyöke

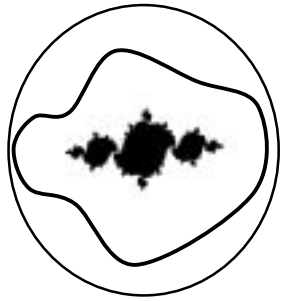
Julia halmaz

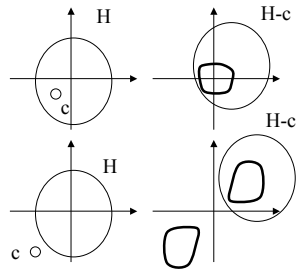


összefüggő

nem összefüggő,
Cantor féle halmaz

Julia halmaz összefüggősége

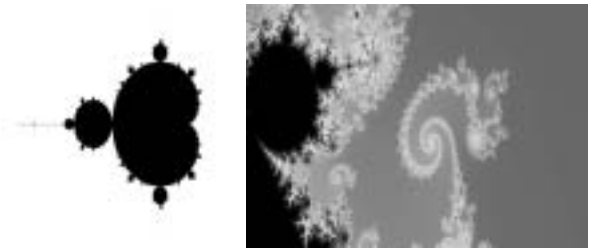




$z_{n+1} = \pm \sqrt{z_n - c}$

Mandelbrot halmaz

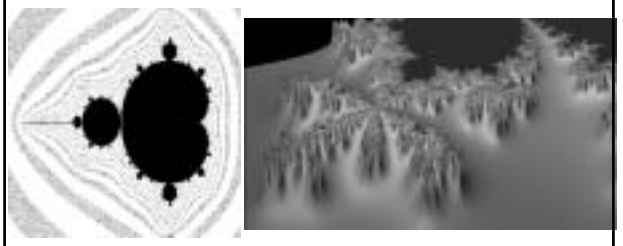
Azon c komplex számok, amelyekre a $z \rightarrow z^2 + c$ Julia halmaza összefüggő



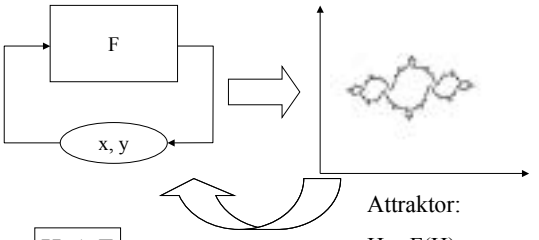
Mandelbrot halmaz, algoritmus

```
MandelbrotDraw ( )
   FOR Y = 0 TO Ymax DO
      FOR X = 0 TO Xmax DO
         ViewportWindow(X,Y → x, y)
         c = x + j y
         z = 0
         FOR i = 0 TO n DO z = z² + c
         IF |z| > "infinity" THEN  WRITE(X,Y, white)
         ELSE                      WRITE(X,Y, black)
      ENDFOR
   ENDFOR
END
```

„Színes" Mandelbrot halmaz



Inverz feladat: IFS modellezés



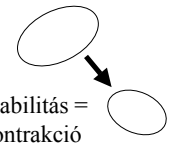
H ⇨ F

F: többértékű lineáris leképzés

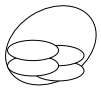
$F = W_1 \vee W_2 \vee \ldots \vee W_n$

$W(x,y) = [ax + by + c, dx + ey + f]$

Stabilitás = kontrakció



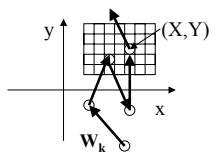
$H = W_1(H) \cup W_2(H) \cup \ldots \cup W_n(H)$



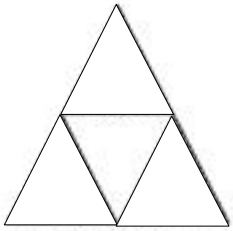
$H = F(H)$

IFS rajzolás: algoritmus

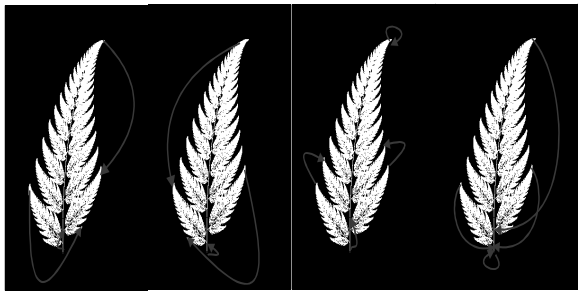
```
IFSDraw ( )
   Legyen [x,y] = [x,y] A₁ + q₁ megoldása a kezdő [x,y]
   FOR i = 0 TO n DO
      IF ClipWindow(x, y)
         WindowViewport(x, y ⇨ X, Y)
         Write(X, Y, color);
      ENDIF
      Válassz k-t pₖ valószínűséggel
      [x,y] = [x,y] Aₖ + qₖ
   ENDFOR
END
```



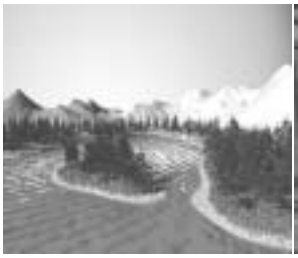
Egyszerű IFS-ek



IFS modellezés



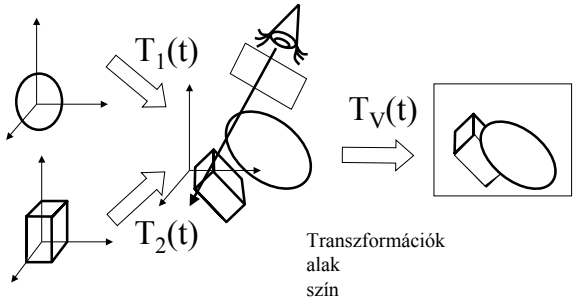
IFS képek



Animáció

Szirmay-Kalos László

Animáció = időfüggés



Transzformációk
alak
szín
megjelenítési attribútumok, stb

Valós idejű animáció

Legalább 15 ciklus másodpercenként

Óra inicializálás (t_{start})
do
    t = Óra leolvasás
    for each object o: modellezési transzf $T_{M,o} = T_{M,o}(t)$
    Nézeti transzformáció: $T_V = T_V(t)$
    Képszintézis
while ($t < t_{end}$)

Nem valós idejű animáció

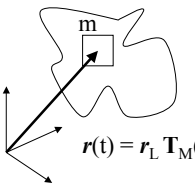
```
t = t_start
do
        for each object o: modellezési transzf T_M,o = T_M,o(t)
        Nézeti transzformáció: T_V = T_V(t)
        Képszintézis, Képtárolás
        t += Δt
while (t < t_end)                                   felvétel
```

```
Óra inicializálás (t_start)                         visszajátszás
do
        t = Óra leolvasás
        Következő kép betöltése
        t += Δt
        while( t > Óra leolvasás ) Várj
while (t < t_end)
```

Valószerű mozgás

- Fizikai törvények:
  - Newton törvény
  - ütközés detektálás és válasz: impulzus megmaradás
- Fiziológiai törvények
  - csontváz nem szakad szét
  - meghatározott szabadságfokú ízületek
  - bőr rugalmasan követi a csontokat
- Energiafelhasználás minimuma

Newton törvény



$\mathbf{F}/m = \frac{d^2}{dt^2}\boldsymbol{r}$

$\boldsymbol{r}(t) = \boldsymbol{r}_L\,\mathbf{T}_M(t)$

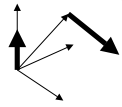
$= \boldsymbol{r}_L \frac{d^2}{dt^2}\mathbf{T}_M(t)$

Az erő rugalmas mechanizmuson keresztül hat, azaz folytonosan változik

$\mathbf{T}_M(t)$ C^2 folytonos

$\mathbf{T}_M(t)$: Mozgástervezés

- Követelmény: C^1/C^2 folytonosság
- Mátrixelemek nem függetlenek
  - Tervezés független paraméterek terében



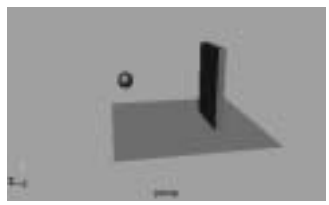
pozíció: px, py, pz
orientáció: α, β, γ
skálázás: sx, sy, sz
} **p**(t)

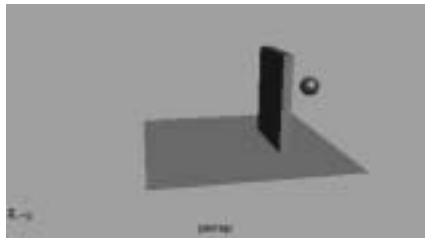
$$\mathbf{T}_M(t) = \begin{bmatrix} sx & & & \\ & sy & & \\ & & sz & \\ & & & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & & \\ -\sin\alpha & \cos\alpha & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & & -\sin\beta & \\ & 1 & & \\ \sin\beta & & \cos\beta & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & \cos\gamma & \sin\gamma & \\ & -\sin\gamma & \cos\gamma & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ px & py & pz & 1 \end{bmatrix}$$

Mozgástervezés a paramétertérben

- **p**(t) elemei C^1/C^2 folytonosak
- **p**(t) elemeinek a definíciója:
  - görbével direkt módon (**spline**)
  - képlettel: **script animation**
  - kulcsokból interpolációval**: keyframe animation**
  - görbével indirekt módon: **path animation**
  - mechanikai modellből az erők alapján: **physical animation**
  - mérésekből: **motion capture animation**

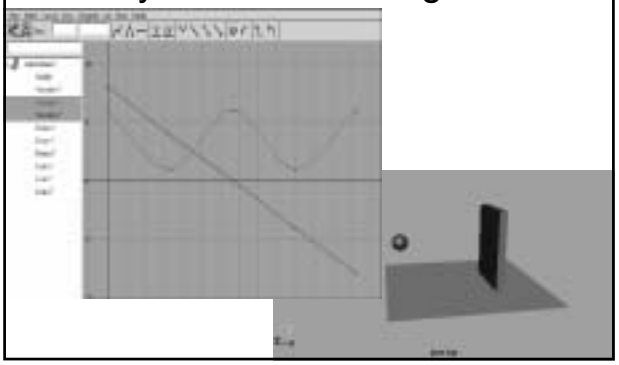
Keyframe animáció



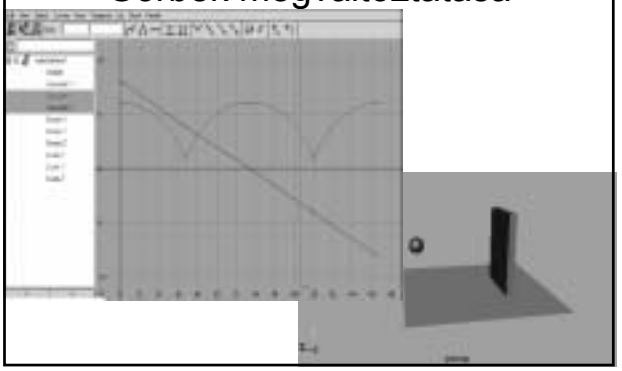


5.key

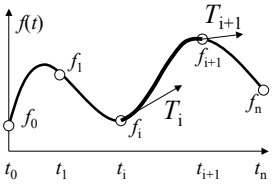
Keyframe animáció görbéi



Görbék megváltoztatása



Interpoláció: 3-d rendű spine



$$f_i(t) = a_i\, t^3 + b_i\, t^2 + c_i t^1 + d_i$$

$$f_i(t_i) = f_i, \qquad f_i(t_{i+1}) = f_{i+1}$$

$$f_i'(t_i) = T_i \qquad f_i'(t_{i+1}) = T_{i+1}$$

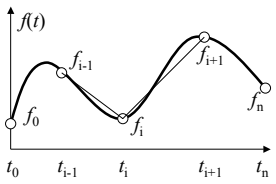
Ismeretlen T_i -k meghatározása:

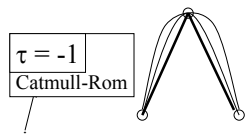
- C^2 folytonosság követelményéből: spline

  $f_i''(t_{i+1}) = f_{i+1}''(t_{i+1})$ + sebesség a kezdő és végpontban

- Tervezési paraméterek alapján: Kohanek-Bartels, Catmull-Rom

Kohanek-Bartels „spline”





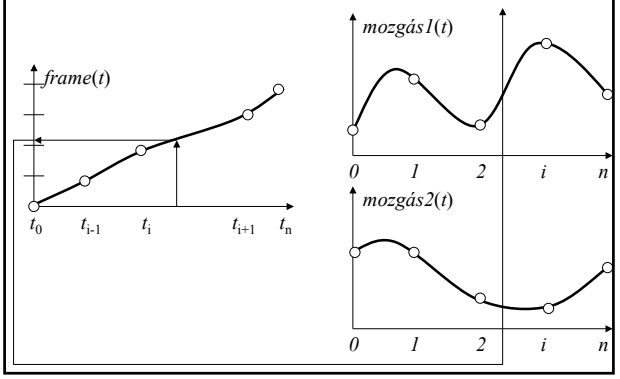
tension paraméter

$$T_i = \frac{1}{2}\left(\frac{f_{i+1} - f_i}{t_{i+1} - t_i} + \frac{f_i - f_{i-1}}{t_i - t_{i-1}} \right) (1-\tau)$$

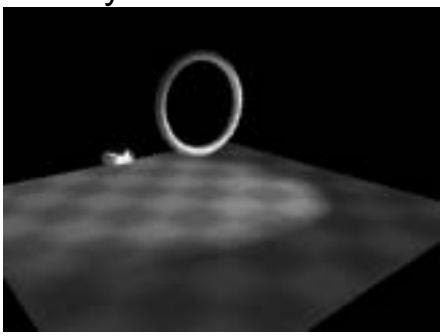
β $(1-\beta)$

β=0.9

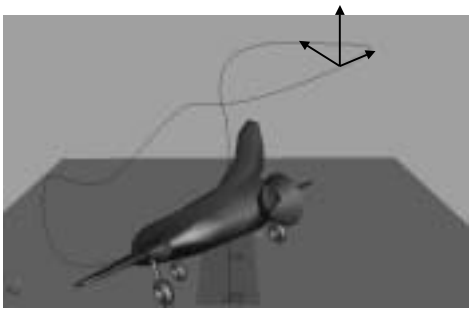
Kinetikai vezérlés



Keyframe animáció



Pálya (path) animáció

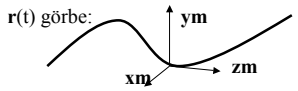


lokális modellezési transzf:

y = fix
z = görbe der.
x = vektor ⊗

t = spline paraméter vagy az ívhossz

Pálya animáció: Transzformáció



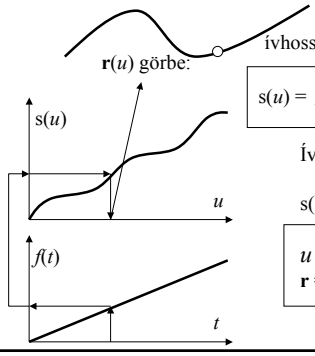
Explicit up vektor
zm = **r**'(t)
xm = **z** ⊗ up
ym = **z** ⊗ **x**

Frenet keretek:
zm = **r**'(t)
xm = **r**'(t) ⊗ **r**''(t)
ym = **z** ⊗ **x**

$$T_M = \begin{pmatrix} \mathbf{xm}^0(t) & 0 \\ \mathbf{ym}^0(t) & 0 \\ \mathbf{zm}^0(t) & 0 \\ \mathbf{r}(t) & 1 \end{pmatrix}$$

A függőleges, amerre az erő hat

Ívhossz szerinti mozgás



$$s(u) = \int \sqrt{(dx/du)^2 + (dy/du)^2 + (dz/du)^2}\, du$$

Ívhossz szerinti mozgás:

$s(u) = f(t)$, például $a \cdot t$

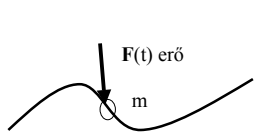
$u = s^{-1}(f(t))$
r = **r**(*u*)

Pálya animáció

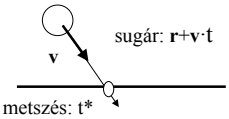
Fizikai animáció

- Erők (gravitáció, turbulencia, stb.)
- Tömeg, tehetetlenségi nyomaték (F = ma)
- Ütközés detektálás (metszéspontszámítás)
- Ütközés válasz
  - rugók, ha közel vannak
  - impulzus megmaradás alapján

Egy kis mechanika



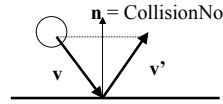
ÜtközésDetektál



Ha t* < dt Collision

```
for( t = 0; t < T; t += dt) {
    F(t) = Erők számítása
    a = F(t)/m
    v += a ·dt
    if ( ÜtközésDetektál )
        ÜtközésVálasz
    r += v ·dt
}
```

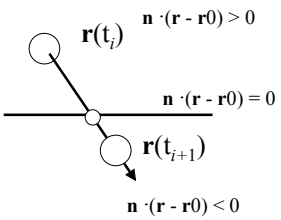
ÜtközésVálasz



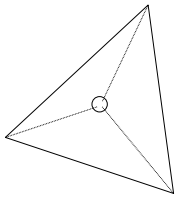
v' = (2(v - v·n) - v) ·bounce

Diszkrét ütközés detektálás

Pont-sík



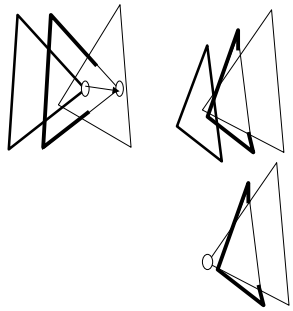
Pont-háromszög



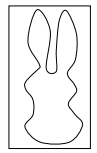
- pont a három oldal jó oldalán van
- baricentrikus koordináták pozitívak
- oldal látószögek összege 360

Ütközés detektálás

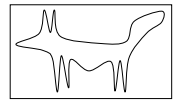
háromszög-háromszög



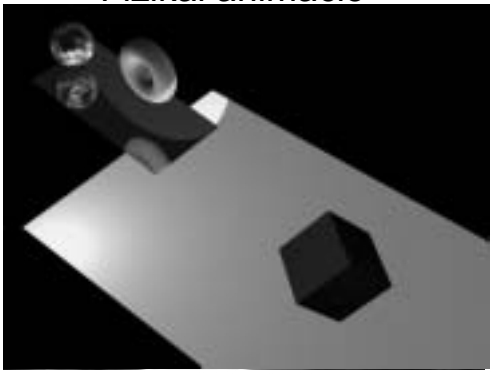
gyorsítás: befoglalók



$O(n^2)$



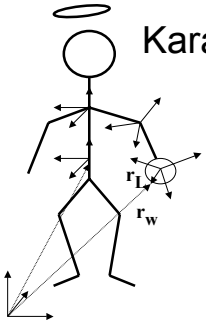
Fizikai animáció



Real-time fizikai animáció

Konyha Zoltán (autoszimulátor)

Karakter animáció



$\mathbf{r_w} = \mathbf{r_L} \cdot \mathbf{R}_{kéz} \cdot \mathbf{T}_{alkar} \cdot \mathbf{R}_{könyök} \cdot \mathbf{T}_{felkar} \cdot \mathbf{R}_{váll} \cdot \mathbf{T}_{gerinc} \cdot \mathbf{T}_{ember}$

homogén koordináta 4-es

Robot példa

T0 = robot előrehalad
glTranslatef(xr, yr, zr);

T1= kar elhelyése
glTranslatef(x, y, z);

T2= forgatás
glRotatef(angle, 1.0f, 0.0f, 0.0f);

T3= skálázás
glScalef(1.0f, 4.0f, 1.0f);

Robot geometria 1

```
void DrawArm(float x, float y, float z, float angle) {
   glPushMatrix();
      glColor3f(1.0f, 0.0f, 0.0f);  // red
      glTranslatef(x, y, z);
      glRotatef(angle, 1.0f, 0.0f, 0.0f);
      glScalef(1.0f, 4.0f, 1.0f);   // 1x4x1 cube
      DrawCube( );
   glPopMatrix();
}

void DrawHead(float x, float y, float z) {
   glPushMatrix();
      glColor3f(1.0f, 1.0f, 1.0f);  // white
      glTranslatef(x, y, z);
      glScalef(2.0f, 2.0f, 2.0f);   // 2x2x2 cube
      DrawCube( );
   glPopMatrix();
}
```

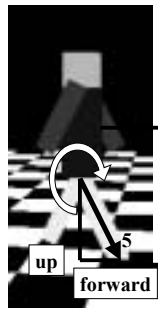
Robot geometria 2

```
void DrawTorso(float x, float y, float z) {
   glPushMatrix();
      glColor3f(0.0f, 0.0f, 1.0f);  // blue
      glTranslatef(x, y, z);
      glScalef(3.0f, 5.0f, 2.0f);   // 3x5x2 cube
      DrawCube( );
   glPopMatrix();
}

void DrawLeg(float x, float y, float z, float angle) {
   glPushMatrix();
      glColor3f(1.0f, 1.0f, 0.0f);  // yellow
      glTranslatef(x, y, z);
      glRotatef(angle, 1.0f, 0.0f, 0.0f);
      glScalef(1.0f, 5.0f, 1.0f);   // 1x5x1 cube
      DrawCube(0.0f, 0.0f, 0.0f);
   glPopMatrix();
}
```

Robot rajzolás + animáció

```
void DrawRobot(float xPos, float yPos, float zPos) {
   glPushMatrix();
      glTranslatef(xPos, yPos, zPos);

      DrawHead(1.0f, 2.0f, 0.0f);
      DrawTorso(1.5f, 0.0f, 0.0f);

      glPushMatrix();
         legAng[0] += dAng[0];
         if (legAng[0]>30 || legAng[0]<-30) dAng[0]*=-1;
         DrawLeg(-0.5f, -5.0f, -0.5f, legAng[0]);
      glPopMatrix();

      … Másik láb, kezek

   glPopMatrix();
}
```

„Inverz kinematika”



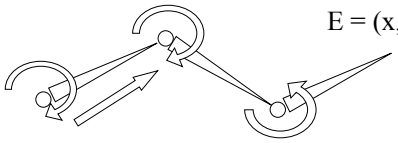
T0 = előrehaladás (forward, up) ???

T2 = forgatás (ang)

A láb (end effektor) földön legyen és ne csúszkáljon

```
forward += 5 * fabs(sin(angNew) - sin(angOld));
up       = 5 * cos(angNew) - 5;
```

Csontváz felépítése



E = (x, y, z, α, β, γ)
end effektor

Állapotvektor = egyes ízületek állapota

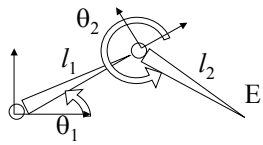
$S = (px, py, pz, \alpha, \beta, \gamma)|_{i=1..n}$

Strukturális összefüggés:

$E = F(S)$

2 szabadságfokú rendszer

$S = (\theta_1, \theta_2)$



$E = (x, y)$

Strukturális összefüggés:

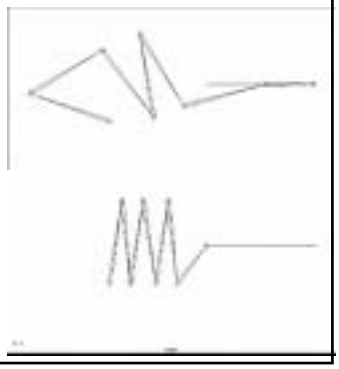
$$x(\theta_1, \theta_2) = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$
$$y(\theta_1, \theta_2) = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$

Forward kinematika

karakter állapot beállítás: S_1
karakter állapot beállítás: S_2
…
karakter állapot beállítás: S_n

S(t) interpolációval

$E = F (S(t))$



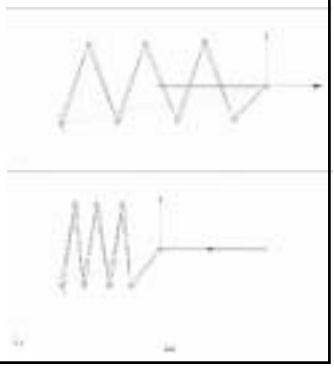
Forward kinematika eredmény

Inverz kinematikával

end effektor beállítás: E_1
end effektor beállítás: E_2
…
end effektor beállítás: E_n

E(t) interpolációval

$S = F^{-1}(E(t))$



Inverz kinematika eredmény

Inverz kinematika: $S = F^{-1}(E)$

Az F strukturális függvény általában nem invertálható analitikusan és az inverz nem egyértelmű: ITERATÍV MEGOLDÁS

- t_0-ra $\mathbf{E}_0$, $\mathbf{S}_0$ ismert
- ha $\mathbf{E}_t$, $\mathbf{S}_t$ ismert, számítsuk $\mathbf{E}_{t+dt} = \mathbf{E}_t + d\mathbf{E}_t$, $\mathbf{S}_{t+dt} = \mathbf{S}_t + d\mathbf{S}$

$$\mathbf{E}_t + d\mathbf{E}_t = \mathbf{F}(\mathbf{S}_t + d\mathbf{S}) = \mathbf{F}(\mathbf{S}_t) + \frac{\partial \mathbf{F}}{\partial \mathbf{S}_1} d\mathbf{S}_1 + \ldots + \frac{\partial \mathbf{F}}{\partial \mathbf{S}_n} d\mathbf{S}_n$$

$$\begin{pmatrix} d\mathbf{E}_{t,1} \\ \ldots \\ d\mathbf{E}_{t,m} \end{pmatrix} = \begin{pmatrix} \frac{\partial \mathbf{F}_1}{\partial \mathbf{S}_1} & \ldots & \frac{\partial \mathbf{F}_1}{\partial \mathbf{S}_n} \\ \ldots & & \ldots \\ \frac{\partial \mathbf{F}_m}{\partial \mathbf{S}_1} & \ldots & \frac{\partial \mathbf{F}_m}{\partial \mathbf{S}_n} \end{pmatrix} \begin{pmatrix} d\mathbf{S}_1 \\ \ldots \\ d\mathbf{S}_n \end{pmatrix}$$

Jacobi-mátrix
m x *n* lineáris egyenletrendszer

m x *n*-es egyenletrendszerek megoldása

$d\mathbf{E}_m = \mathbf{J}_{mxn} \cdot d\mathbf{S}_n$ $\quad n > m$

$\mathbf{J}^T_{nxm} \cdot d\mathbf{E}_m = \mathbf{J}^T_{nxm} \mathbf{J}_{mxn} \cdot d\mathbf{S}_n$

$$(\mathbf{J}^T_{nxm} \mathbf{J}_{mxn})^{-1} \mathbf{J}^T_{nxm} \cdot d\mathbf{E}_m = d\mathbf{S}_n$$

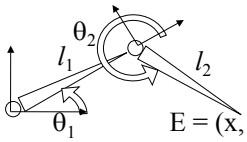
$\mathbf{J}^+ = \mathbf{J}_{nxm}$ pseudo inverze:

a lehetséges megoldásokból egyet választ, amely az állapotvektor változási sebességét minimalizálja

Inverz kinematika megoldás

E = $\mathbf{E}_0$, **S** = $\mathbf{S}_0$
for(t = t0; t < T; t += dt) {
    S alapján a transzformációs mátrixok előállítása
    Képszintézis
    J(**S**) Jacobi mátrix számítása
    $\mathbf{J}^+$ = **J** pseudo-inverze
    E(t+dt) interpolációval (keyframe)
    d**E** = **E**(t+dt) - **E**(t)
    d**S** = $\mathbf{J}^+ \cdot$ d**E**
    S += d**S**
}

Példa: 2 szabadságfokú rendszer



$x(\theta_1,\theta_2) = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$
$y(\theta_1,\theta_2) = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$

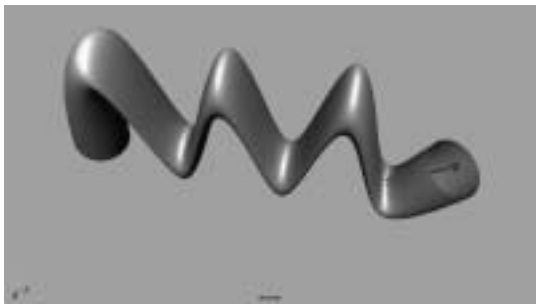
E = (x, y)

$S = (\theta_1,\theta_2)$

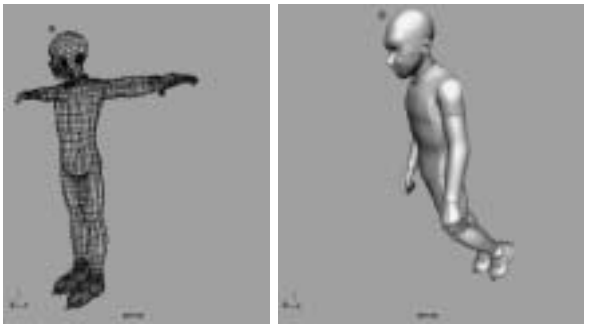
J(θ_1, θ_2) Jacobi mátrix

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} -l_1 \sin\theta_1 - l_2 \sin(\theta_1 + \theta_2) & -l_2 \sin(\theta_1 + \theta_2) \\ l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) & l_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \begin{bmatrix} d\theta_1 \\ d\theta_2 \end{bmatrix}$$

Bőrözés

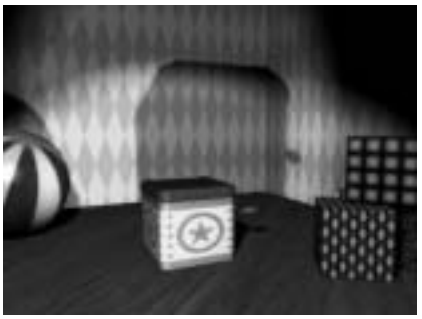


Példa: sétáló mozgás



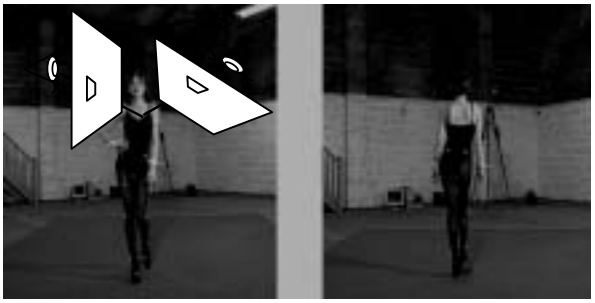
Teljes,ciklikus séta

Egyszerű karakteranimáció

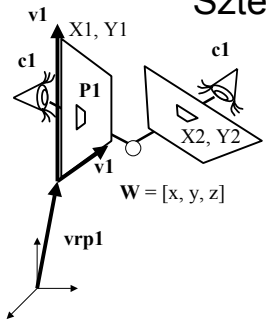


Minimális energiájú mozgás

Motion Capture animáció



Sztereo látás

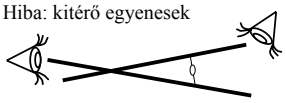


P = **vrp** + X/XMAX **u** + Y/YMAX **v**

c1 + (**P1**-**c1**) · t1 = [x, y, z]
c2 + (**P2**-**c2**) · t2 = [x, y, z]

6 egyenlet, 5 ismeretlen (x, y, z, t1, t2)

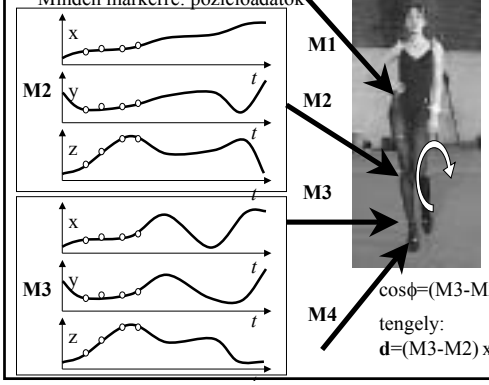
Hiba: kitérő egyenesek



c, u, v, vrp meghatározása: kamera kalibráció

Motion capture eredmény

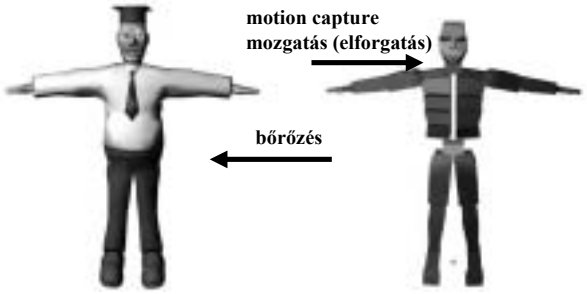
Minden markerre: pozicióadatok



$\cos\phi=(M3-M2)^0 \cdot (M2-M1)^0$

tengely:
d=(M3-M2) x (M2-M1)

Motion Capture adatok alkalmazása

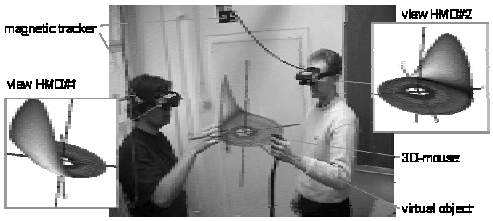


Czuczor Szabolcs, Czékmány Balázs, Aszódi Barnabás: MikroProfesszor

Motion capture animációk

Czuczor Szabolcs, Czékmány Balázs, Aszódi Barnabás: Mikro Professzor

Augmentált valóság

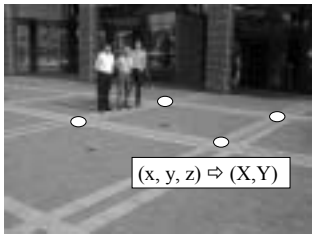


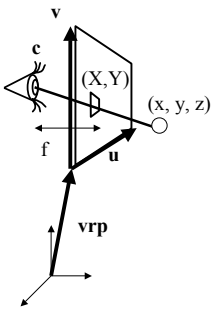
Feladat: felhasználók követése,
kameraparaméterek meghatározása

Valósági és virtuális világok képi illesztése

Blaskó Gábor

Kamera regisztráció





Ismeretlenek: **vrp**, f, **u**, **v**
3 + 1+2+2 = 8

Kompozitálás

Blaskó Gábor