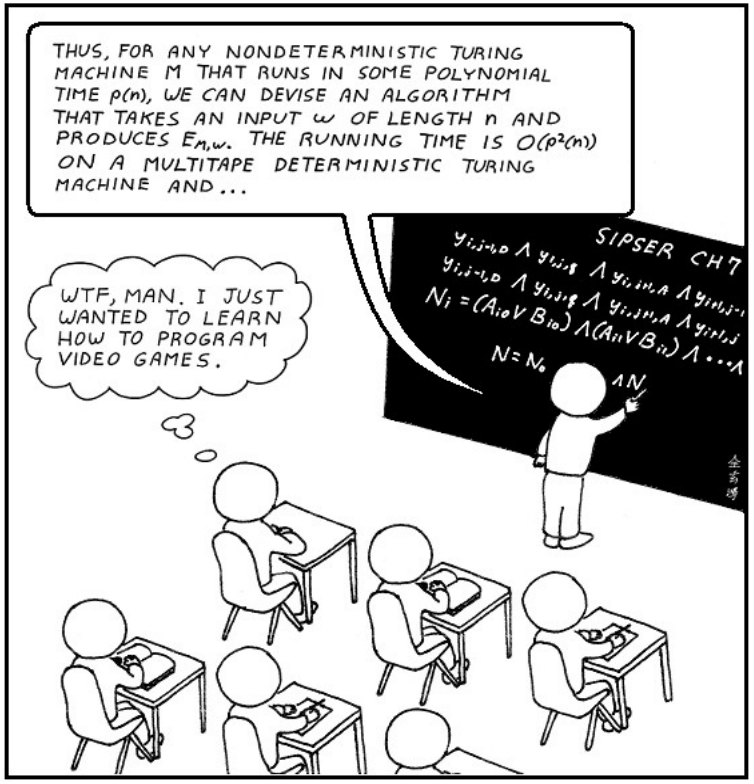
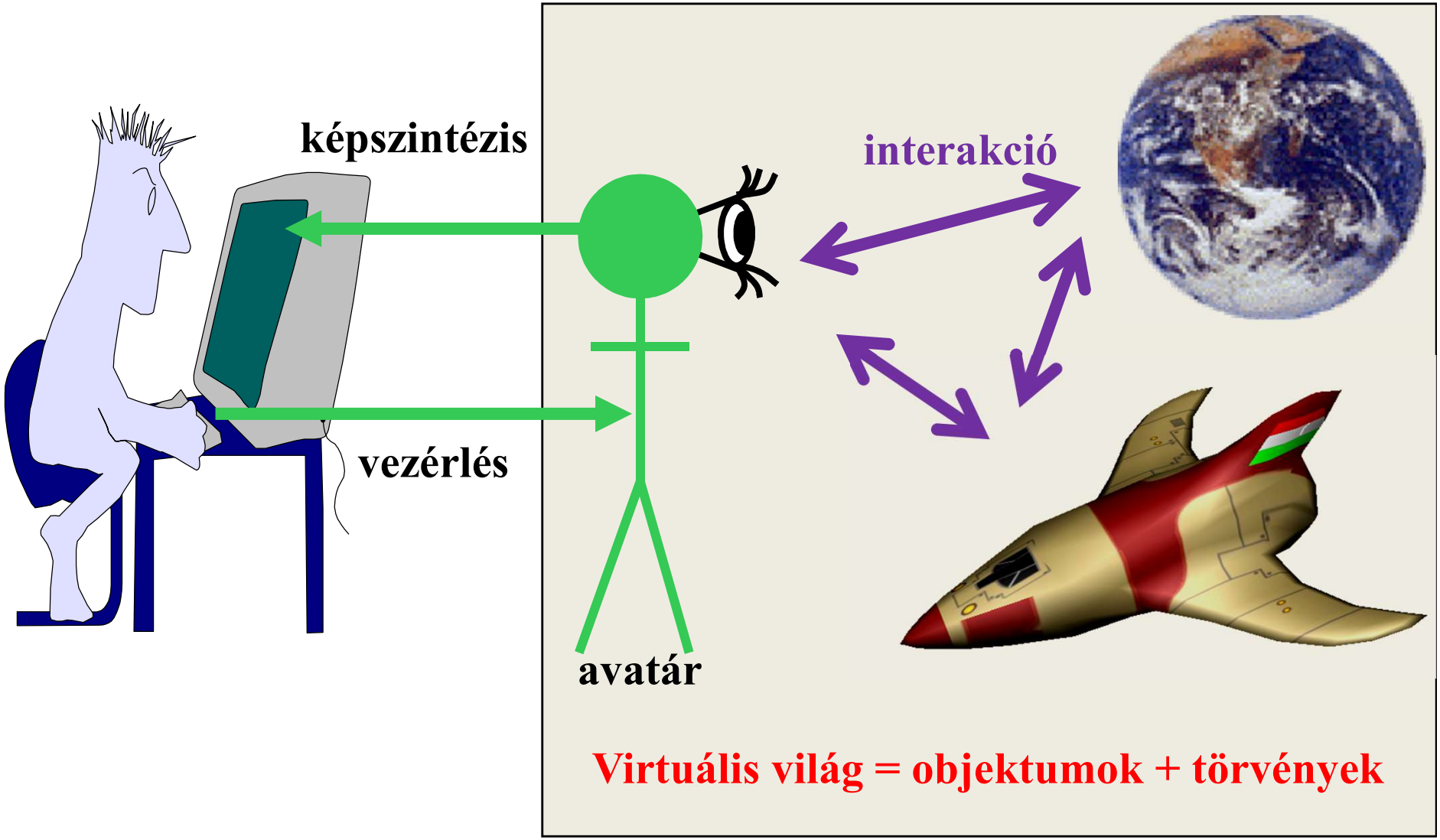
Játékfejlesztés



Szirmay-Kalos László

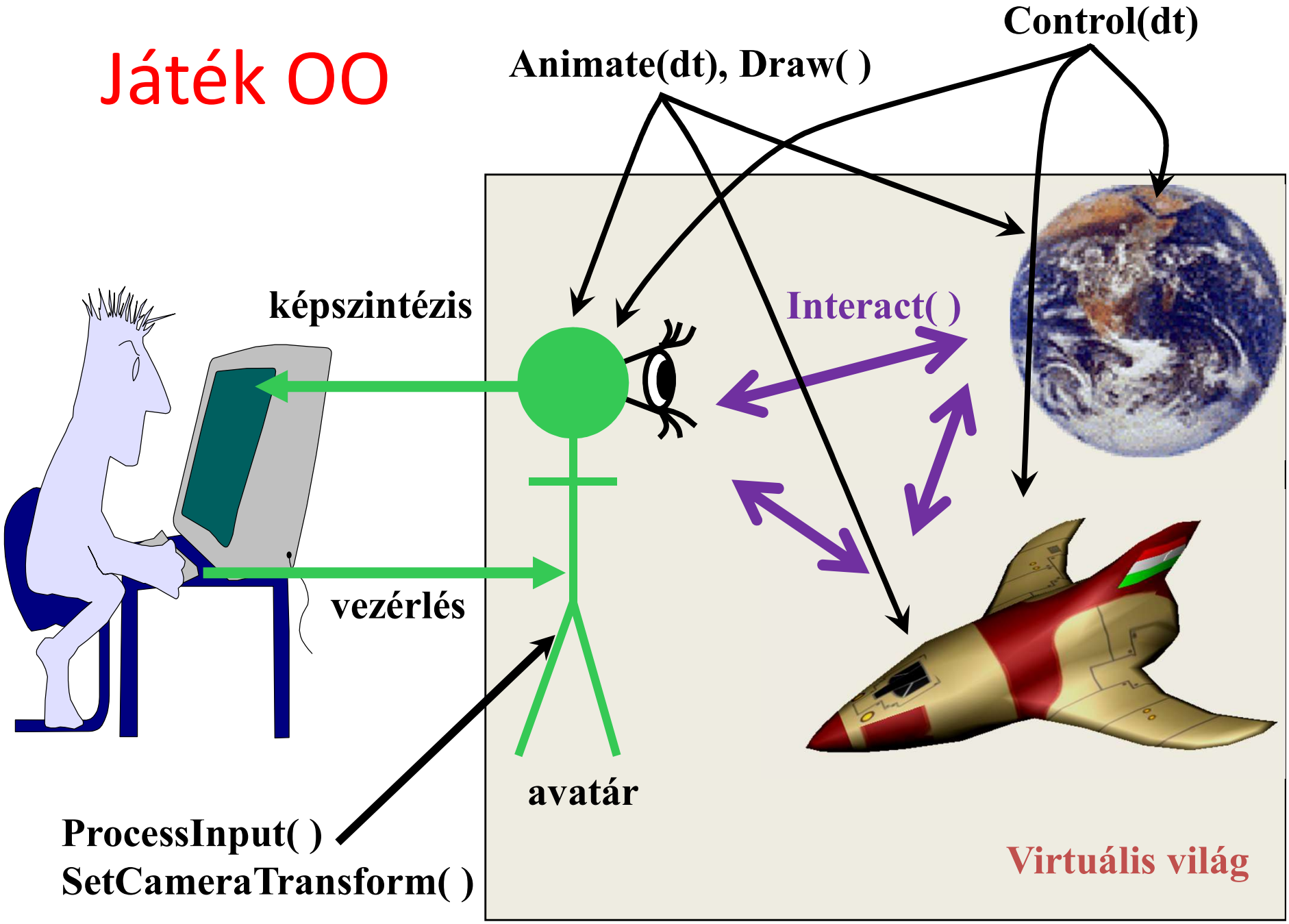
Virtuális valóság



Játékok feladatai

- Képszintézis az avatár nézőpontjából
- Az avatár vezérlése a beviteli eszközökkel (keyboard, mouse, Wii, gépi látás, Kinect, stb.)
- Az „intelligens” objektumok vezérlése (AI)
- A fizikai világ szimulációja

Játék OO


Control(dt)
Animate(dt), Draw( )
képszintézis
Interact( )
vezérlés
avatár
ProcessInput( )
SetCameraTransform( )
Virtuális világ

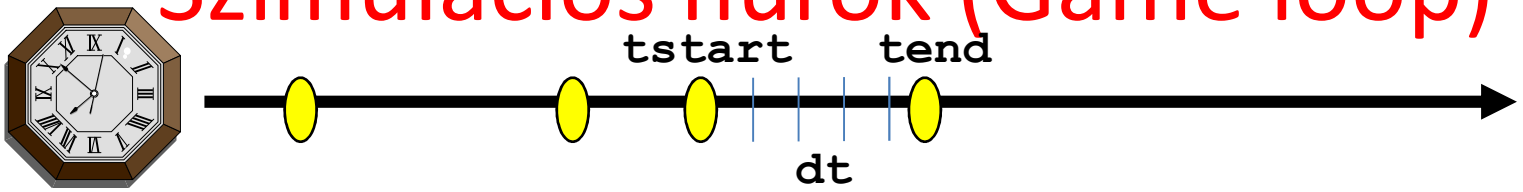
Játékobjektum (GameObject)

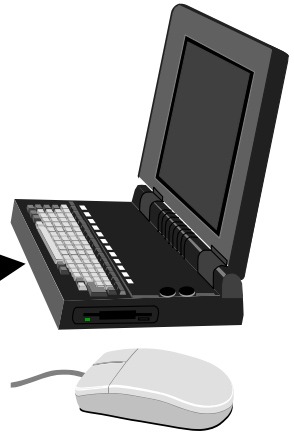
```
class GameObject {
protected:
   Shader *   shader;
   Material * material;
   Texture *  texture;
   Geometry * geometry;
   vec3 pos, velocity, acceleration;
public:
   GameObject(Shader* s, Material* m,
              Texture* t, Geometry* g) { … }
   void Control(float dt) { }
   void Interact(GameObject * o) { }
   void Animate(float dt) { }
   void Draw(RenderState state) { }
};
```

Virtuális világ

```
std::vector<GameObject *> objects;
```

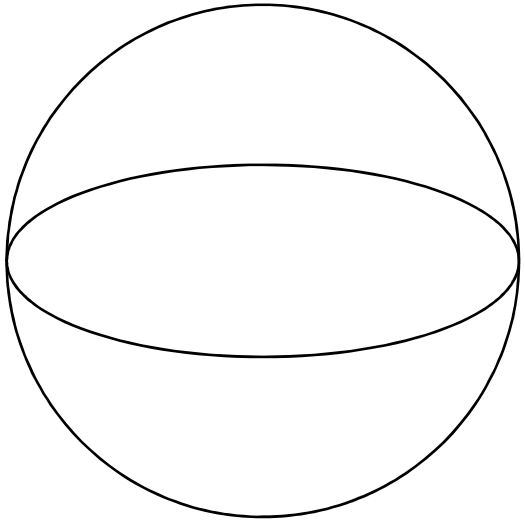
Szimulációs hurok (Game loop)



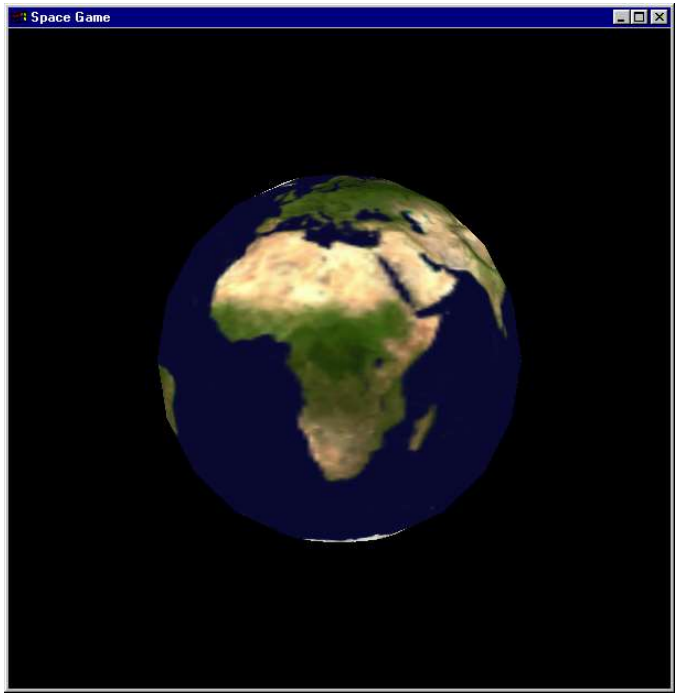


```
void onIdle ( ) {    // idle call back
    static float tend = 0;
    float tstart = tend;
    tend = glutGet(GLUT_ELAPSED_TIME)/1000.0f;
    avatar->ProcessInput( );
    for(float t = tstart; t < tend; t += dt) {
        float Dt = min(dt, tend - t);
        for (GameObject * obj : objects) obj->Control(Dt);
        for (GameObject * obj : objects) obj->Animate(Dt);
    }
    glutPostRedisplay();
}
void onDisplay(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    avatar->SetCameraTransform(state);
    for (GameObject * obj : objects) obj->Draw(state);
    glutSwapBuffers( );
}
```

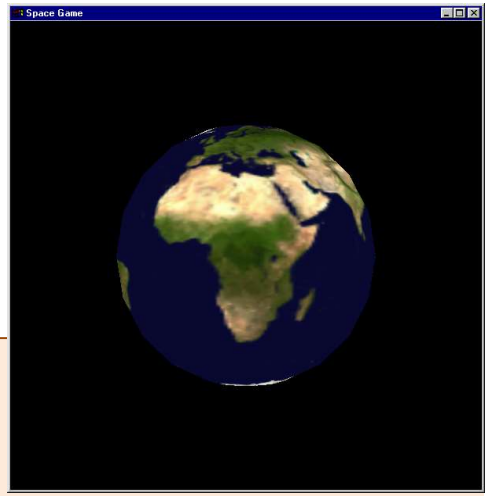
Bolygó: Planet



- Geometria: gömb
- Textúra
- Fizikai vagy
  - Tájékozódik majd követi a gravitációs törvényt
- <u>Képletanimáció:</u>
  - „beégetett pálya”
  - Többiek érdektelenek
  - Nincs respektált törvény

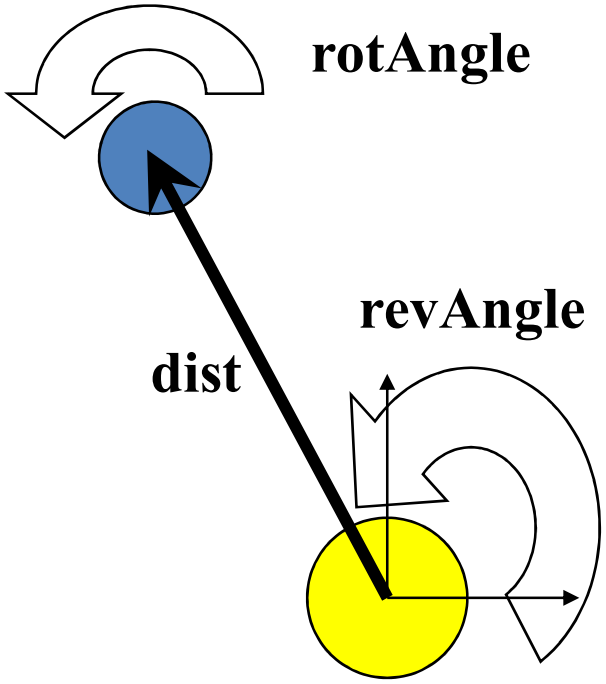


Planet class



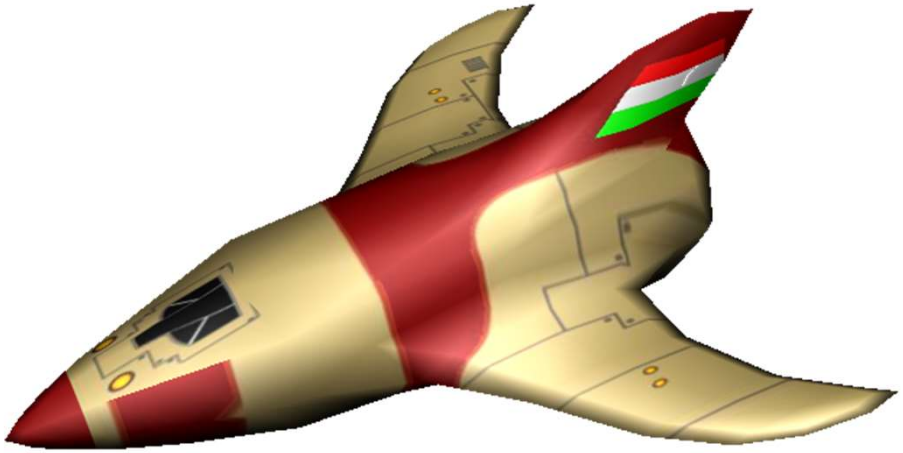

```
class Planet : public GameObject {
   float rotAngle; // animation state
   float rotSpeed; // animation parameter
public:
   Planet(Gouraud* s, Diffuse* m, Texture* t, Sphere* s)
   : GameObject(s, m, t, s) { rotAngle = 0; rotSpeed = …;}

   void Animate(float dt) { rotAngle += rotSpeed * dt; }

   void Draw(RenderState state) {
      state.M = Rotate(rotAngle, 0, 0, 1)
      state.Minv = Rotate(-rotAngle, 0, 0, 1)
      state.material = material;
      state.texture = texture;
      shader->Bind(state);
      geometry->Draw();
   }
};
```

A Föld kering a Nap körül



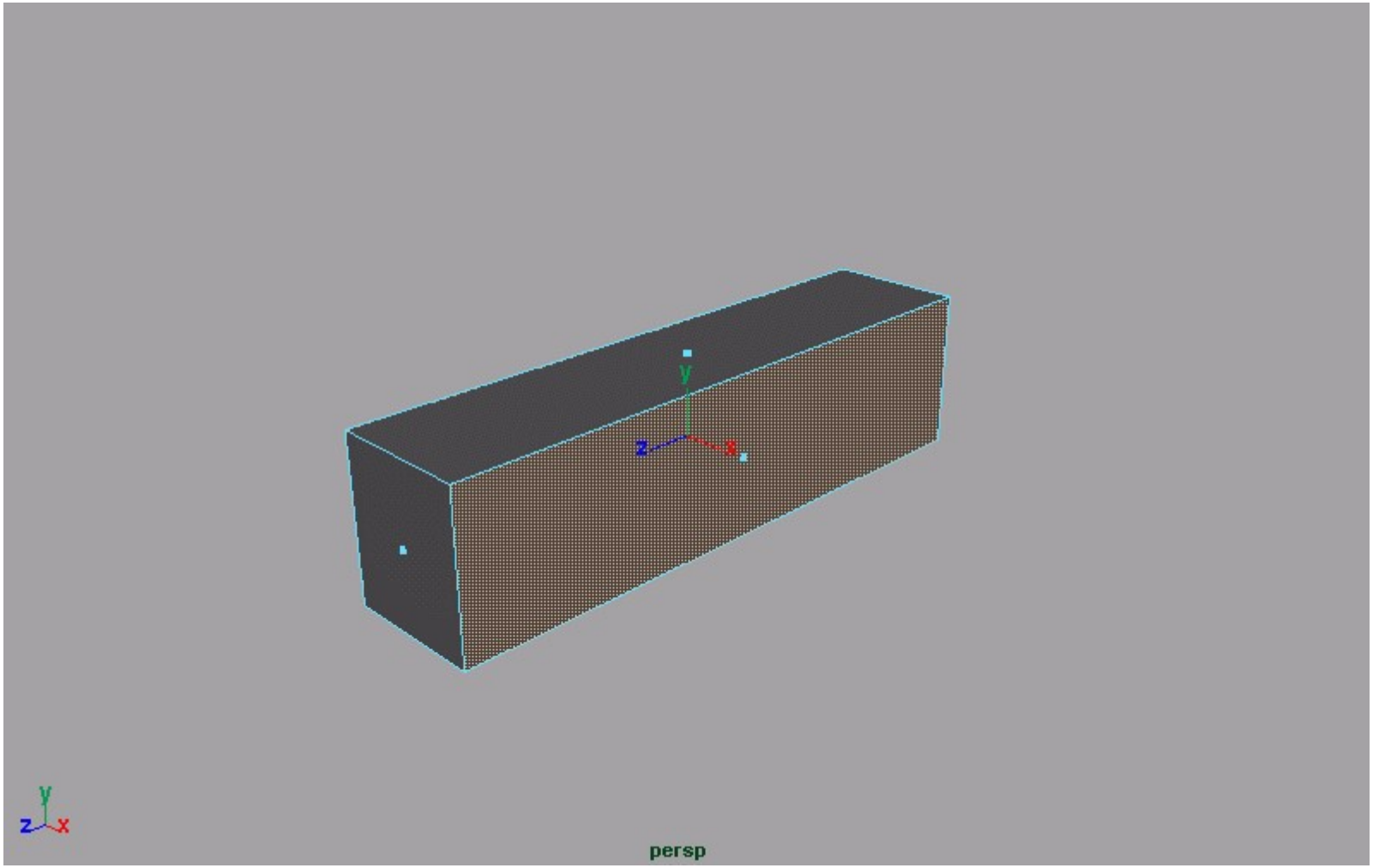

```
void Planet::Animate(float dt){
    rotAngle += rotSpeed * dt;
    revAngle += revSpeed * dt;
}

void Planet::Draw(RenderState state) {
    state.M = Rotate(rotAngle,0,0,1)*
              Translate(dist, 0, 0) *
              Rotate(revAngle,0,0,1);
    state.Minv = …;
    state.material = material;
    state.texture = texture;
    shader->Bind(state);
    geometry->Draw();
}
```

Az űrhajó

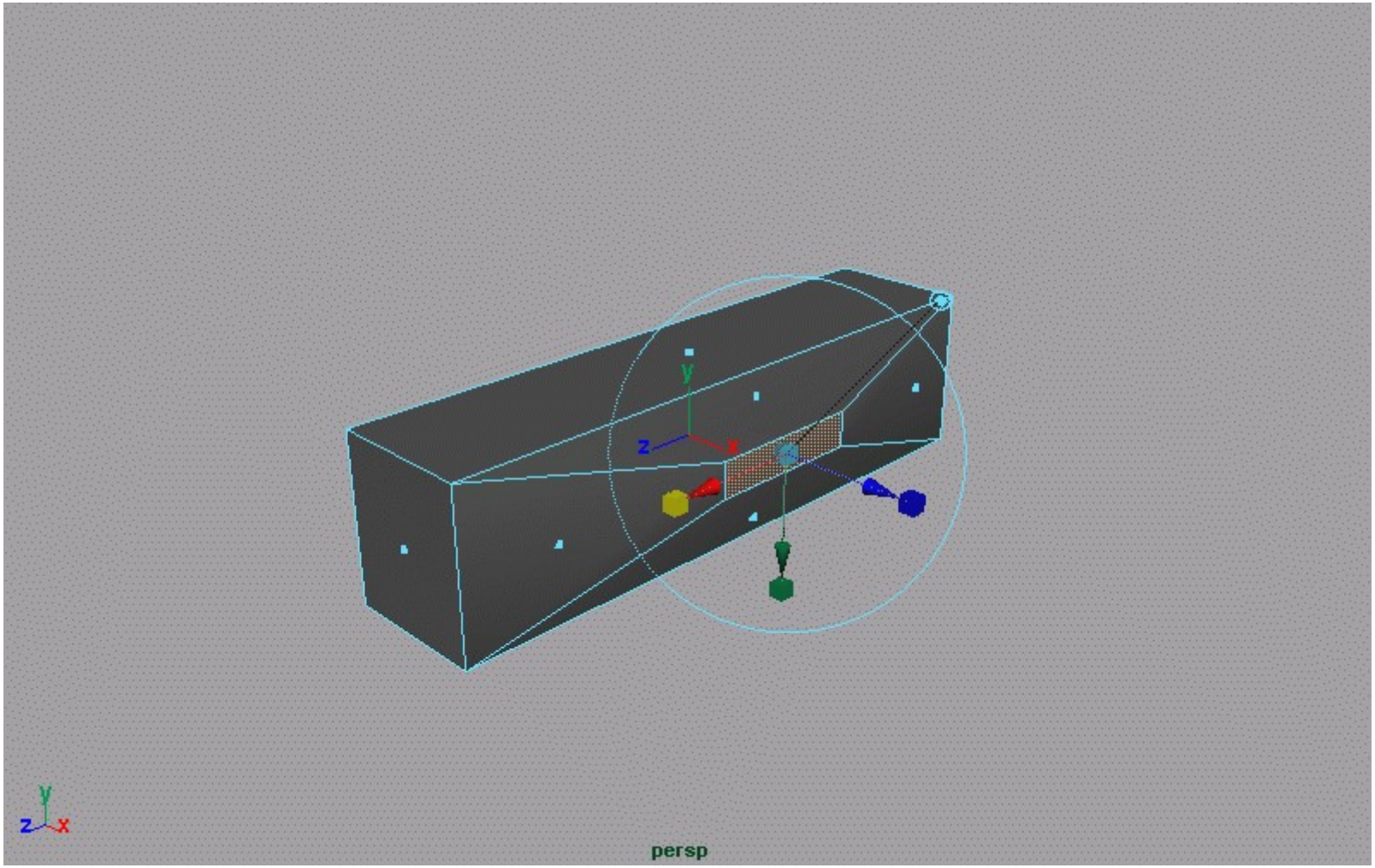


- Komplex geometria
  - négyszögháló
- Komplex textúra
- Fizikai animáció
  - erők (gravitáció, rakéták)
  - ütközések
- Viselkedés (AI)
  - A rakéták vezérlése
    - Ütközés elkerülés, avatártól menekülés, avatár üldözése

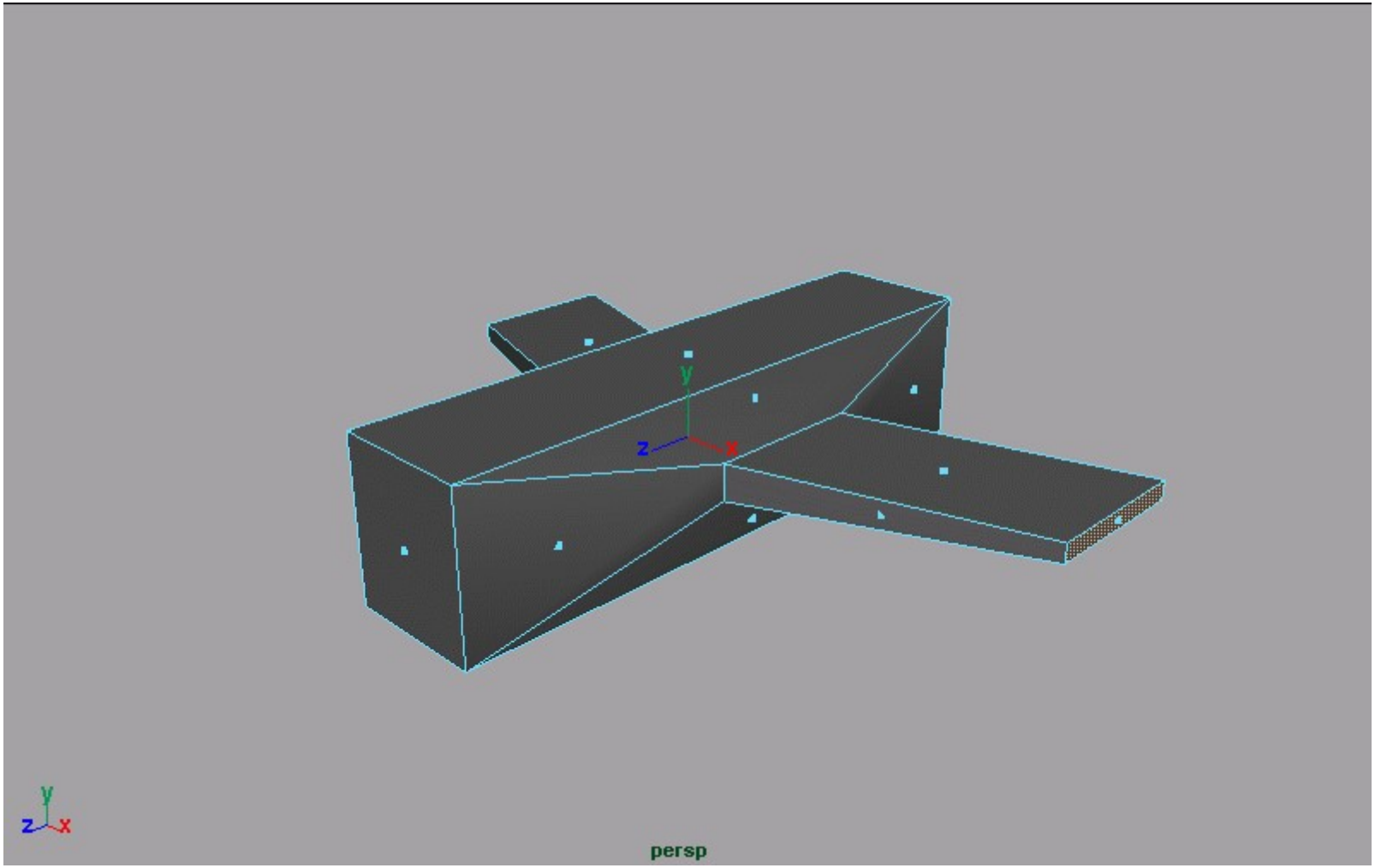
Űrhajó geometria



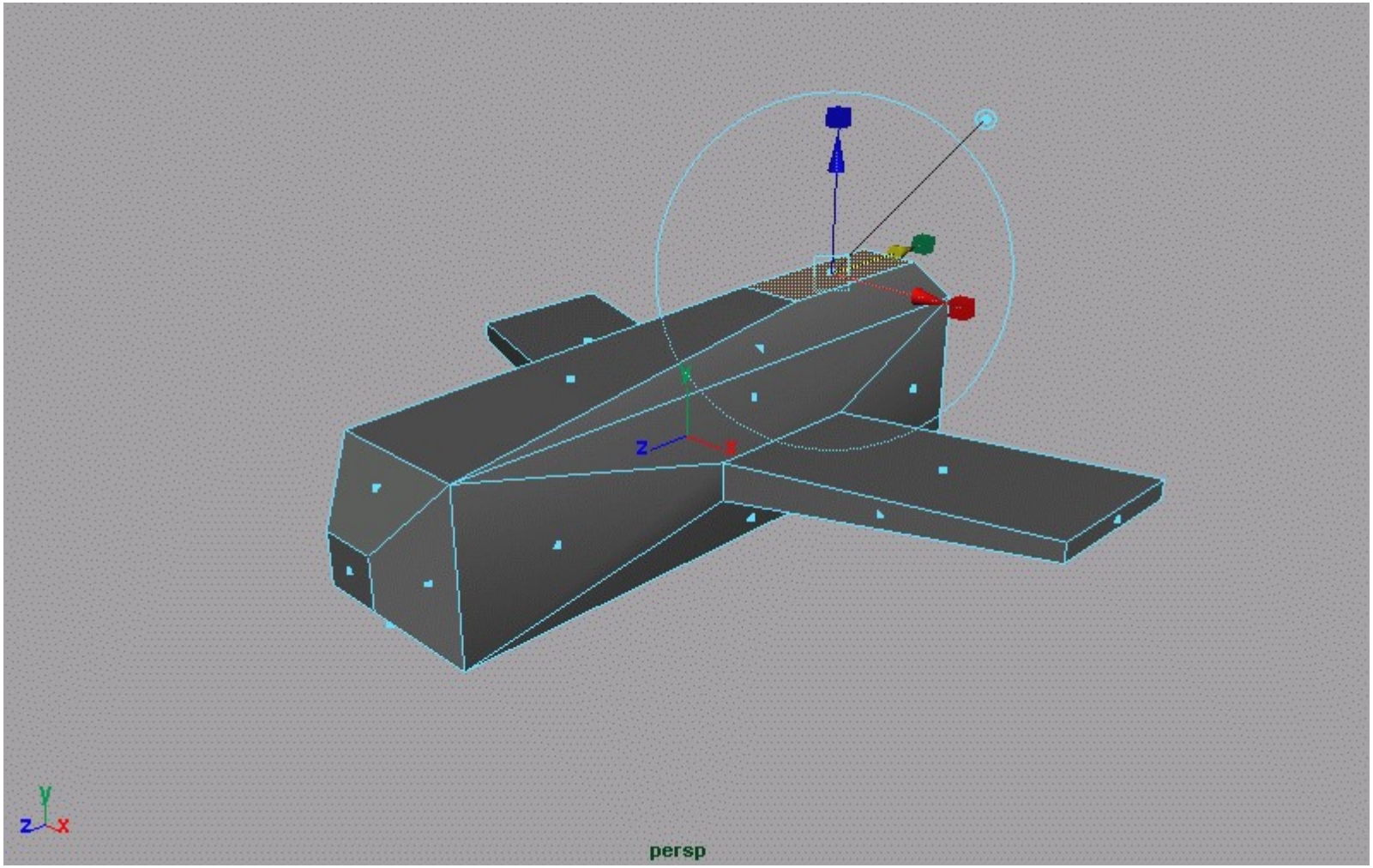
Űrhajó geometria



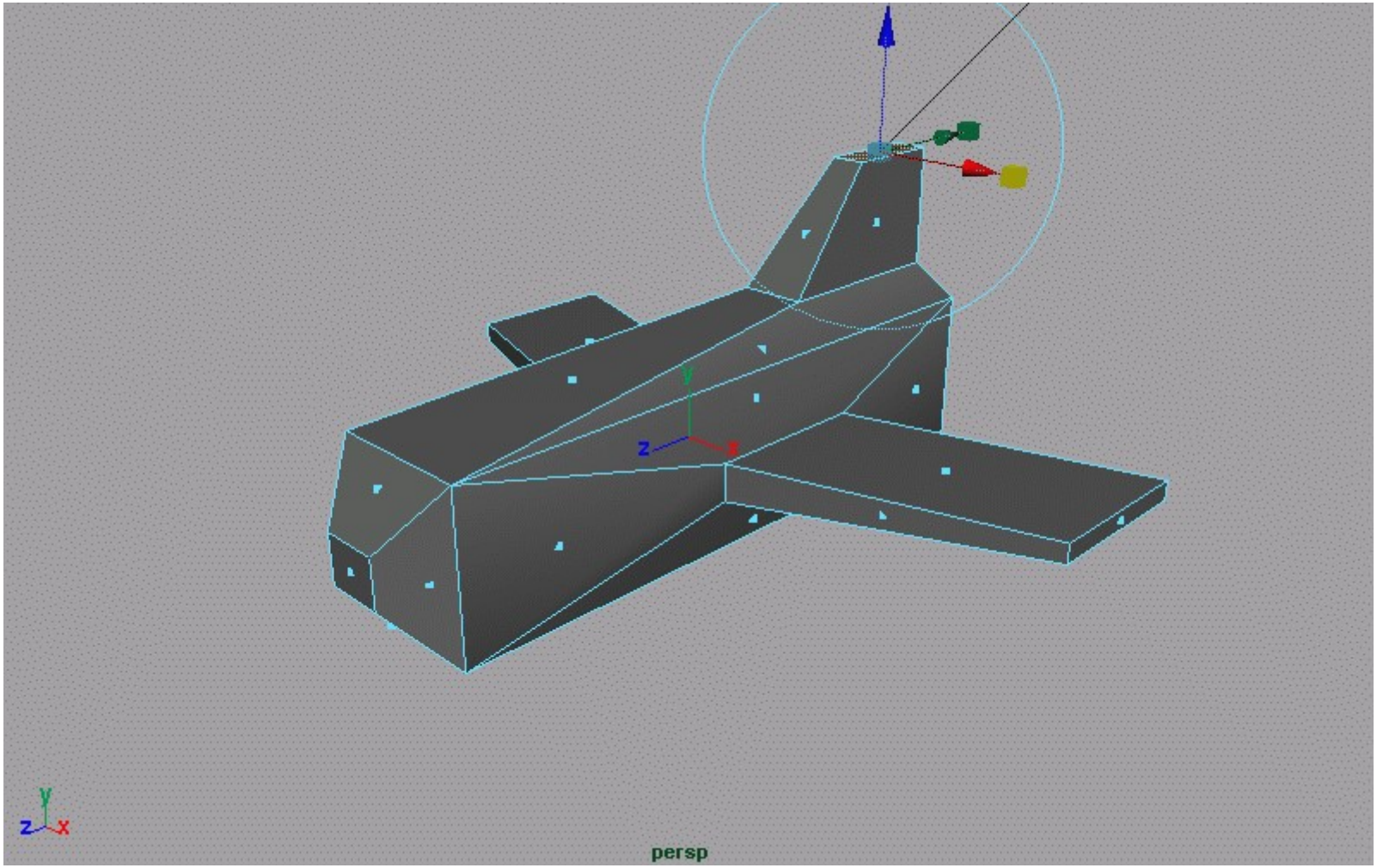
Űrhajó geometria



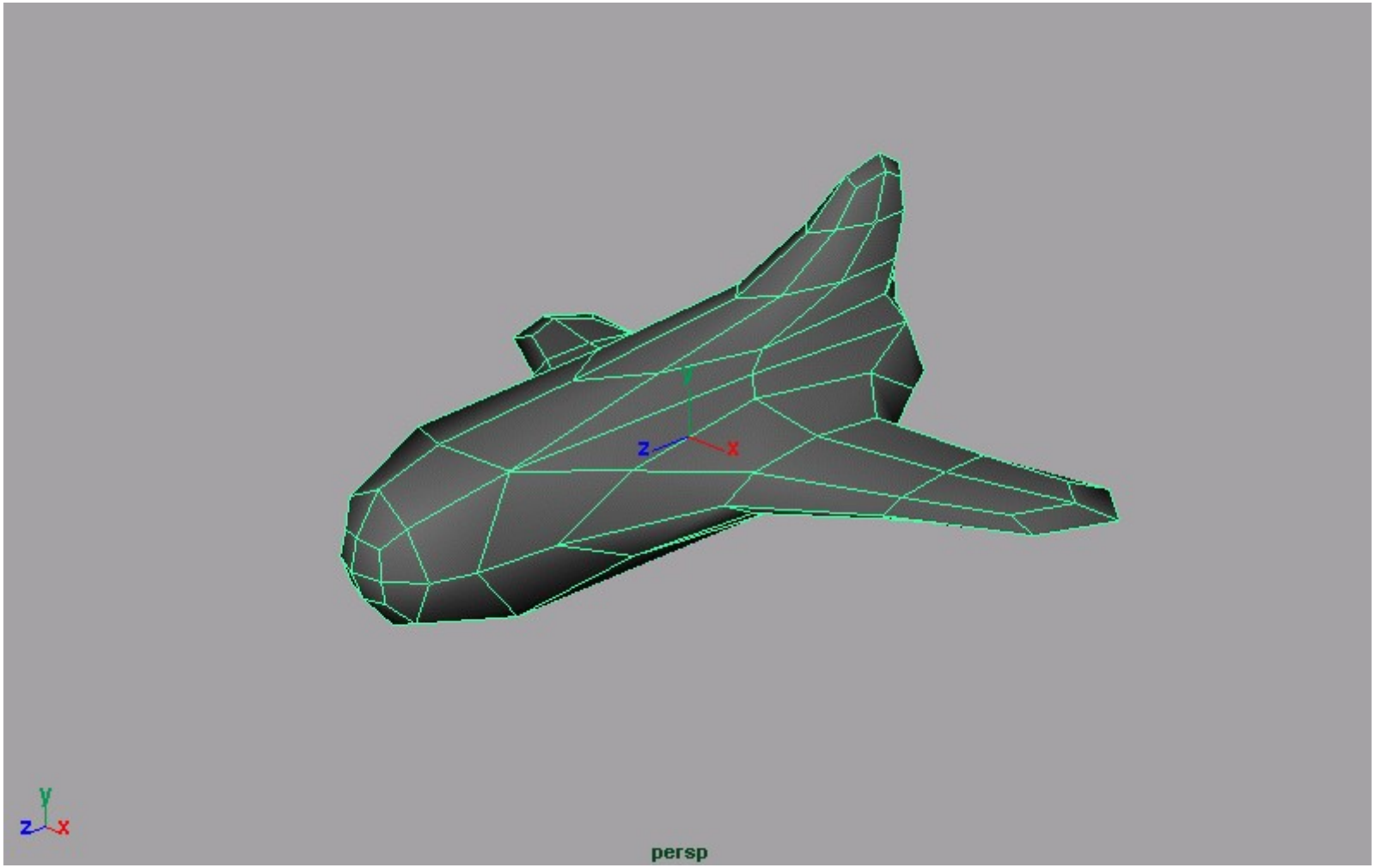
Űrhajó geometria



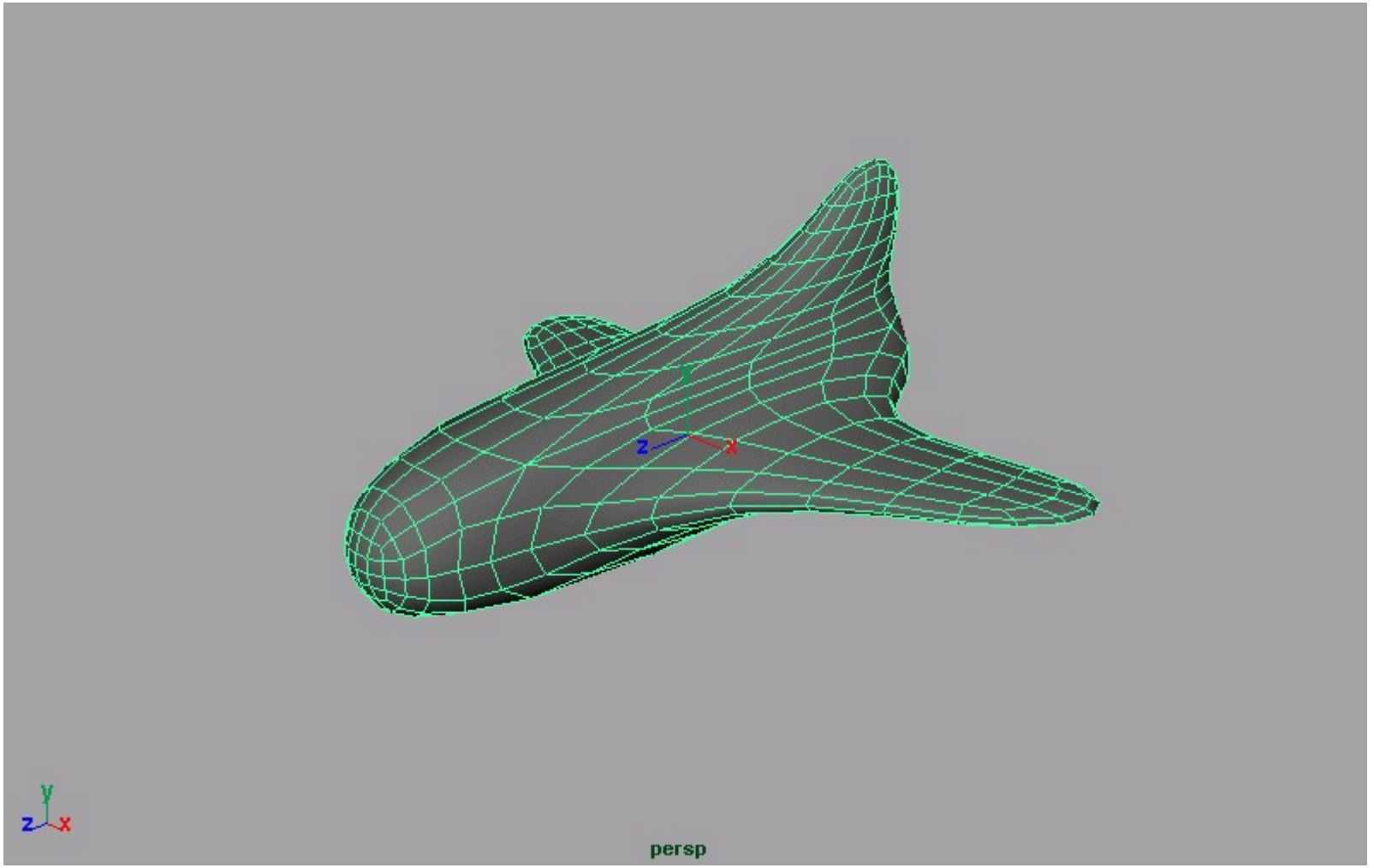
Űrhajó geometria



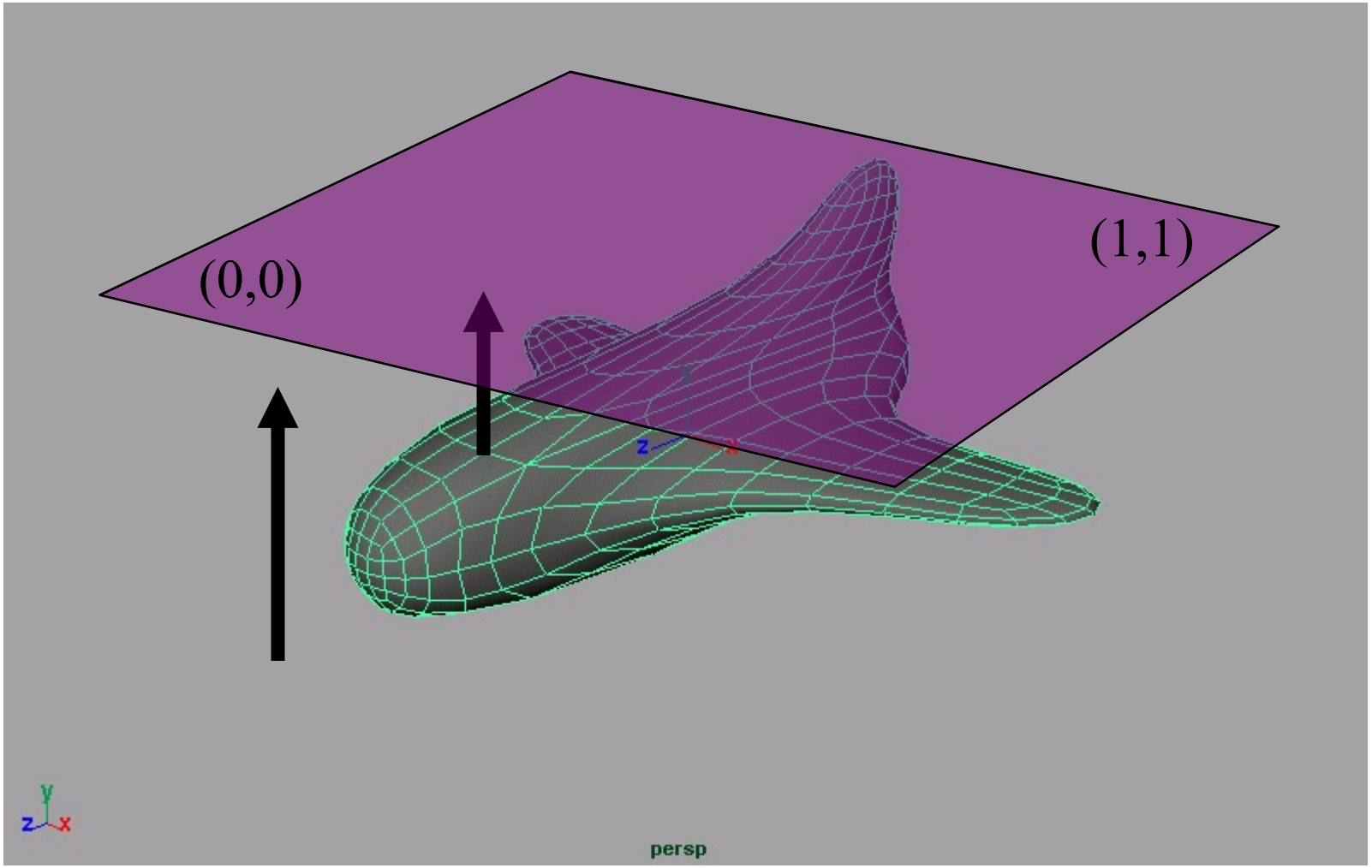
Űrhajó geometria



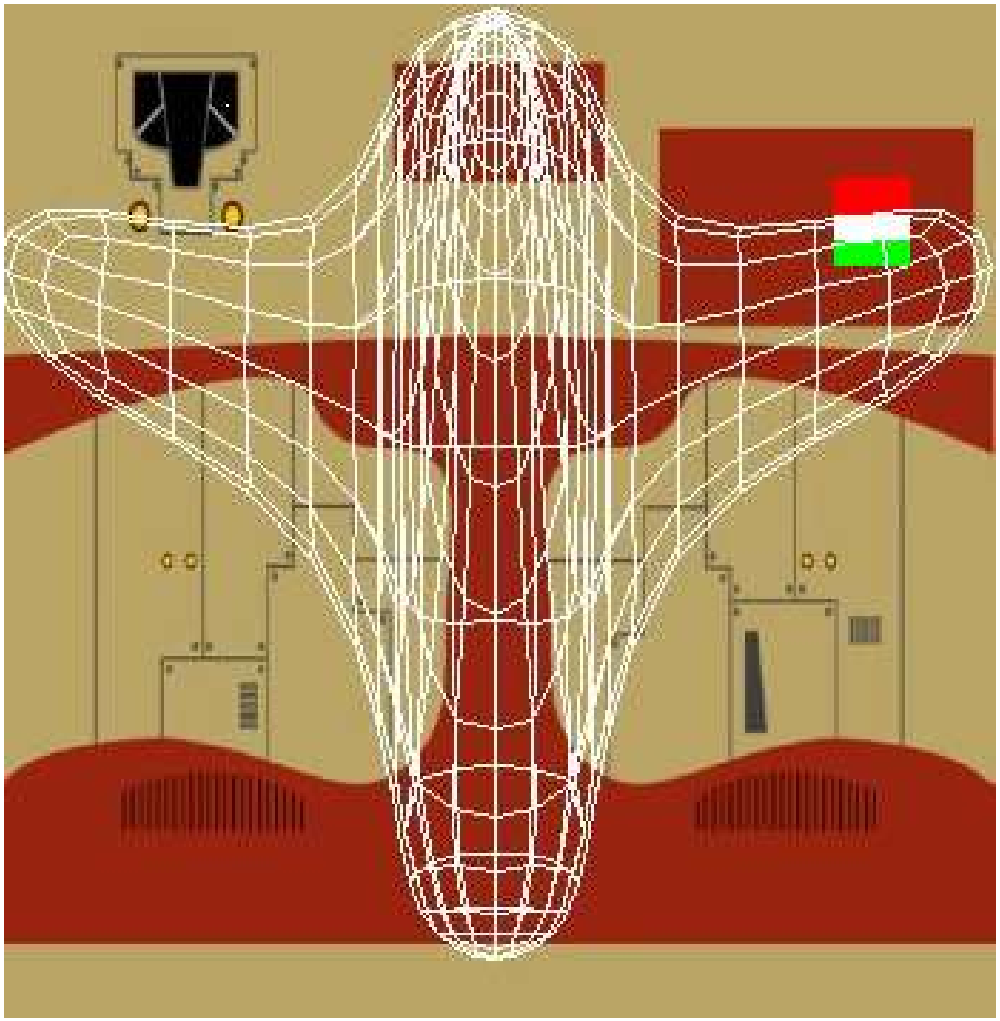
Űrhajó geometria



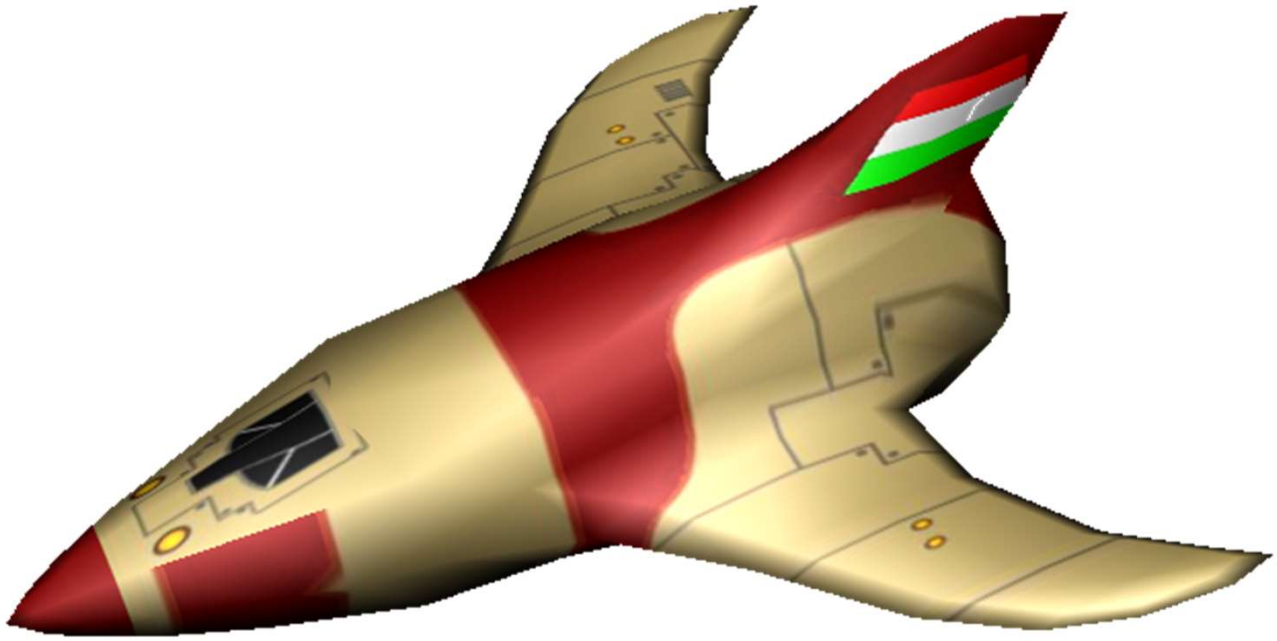
Textúrához paraméterezés



Textúrához paraméterezés



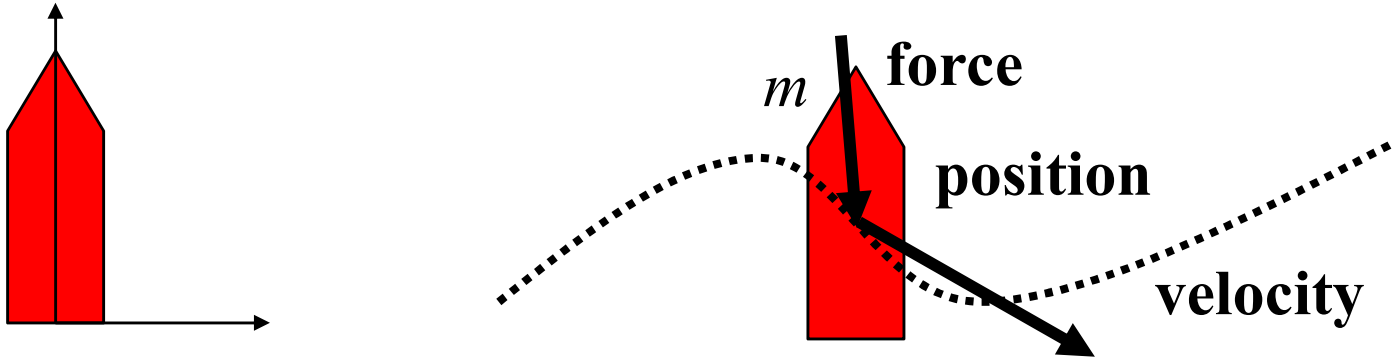
Textúrázott űrhajó



OBJ formátumban

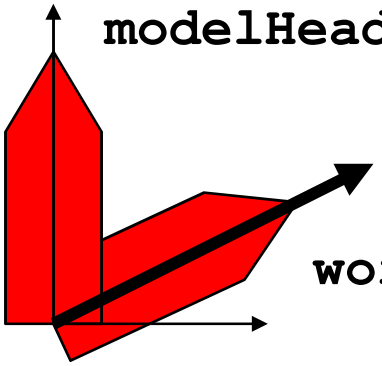
```
v -0.708698 -0.679666 2.277417
v  0.708698 -0.679666 2.277417
v -0.735419  0.754681 2.256846
...
vt 0.510655 0.078673
vt 0.509594 0.070000
vt 0.496429 0.079059
...
vn -0.843091  0.000000 0.537771
vn -0.670151 -0.543088 0.505918
vn -0.000000 -0.783747 0.621081
…
f 65/1/1 37/2/2   62/3/3   61/4/4
f 70/8/5 45/217/6 67/218/7 66/241/8
f 75/9/9 57/10/10 72/11/11 71/12/12
...
```

Animate: Newton mozgástörvényei




```
void Ship :: Animate( float dt ) {
    acceleration = force/m;
    velocity += acceleration * dt;
    pos += velocity * dt;
}

void Ship :: Draw(RenderState state) {
    state.M = Translate(pos.x, pos.y,pos.z);
    shader->Bind(state);
    geometry->Draw();
}
```

Orientáció beállítása

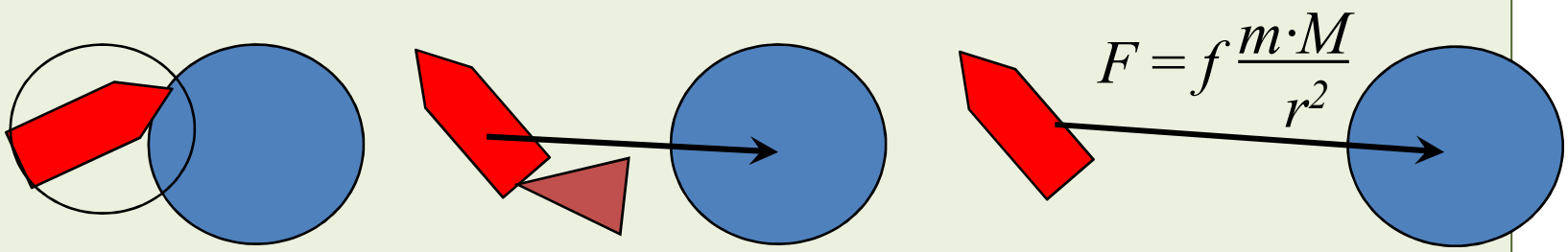



```
worldHead = velocity.normalize();
```

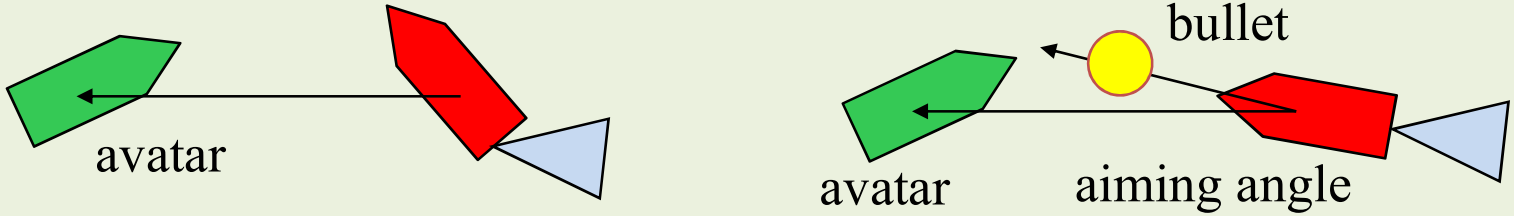
```
void Ship :: Draw(RenderState state) {
  vec3 modelHead( 0, 0, 1 );
  vec3 worldHead = velocity.normalize();
  vec3 rotAx = cross(modelHead, worldHead);

  float cosRotAng = dot(worldHead, modelHead);
  float rotAng = acos(cosRotAngle);
  state.M = Rotate(rotAng, rotAx.x, rotAx.y, rotAx.z)*
            Translate(pos.x, pos.y, pos.z);
  shader->Bind(state);
  geometry->Draw();
}
```

Ship :: Control és Interact

```
void Ship :: Control( float dt ) {
    force = vec3(0, 0, 0);
    for (GameObject * obj : objects) {
        if (dynamic_cast<Planet*>(obj)) {
```

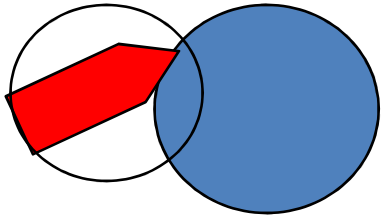



```
        }
        if (dynamic_cast<Avatar*>(obj)) {
```



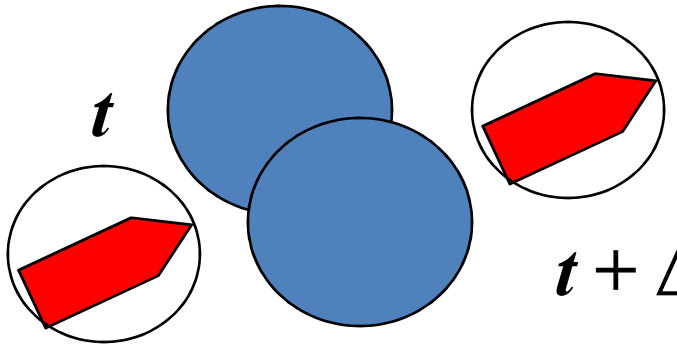

```
        }
    }
}
```

Ütközésdetektálás: lassú objektumok



adott *t*

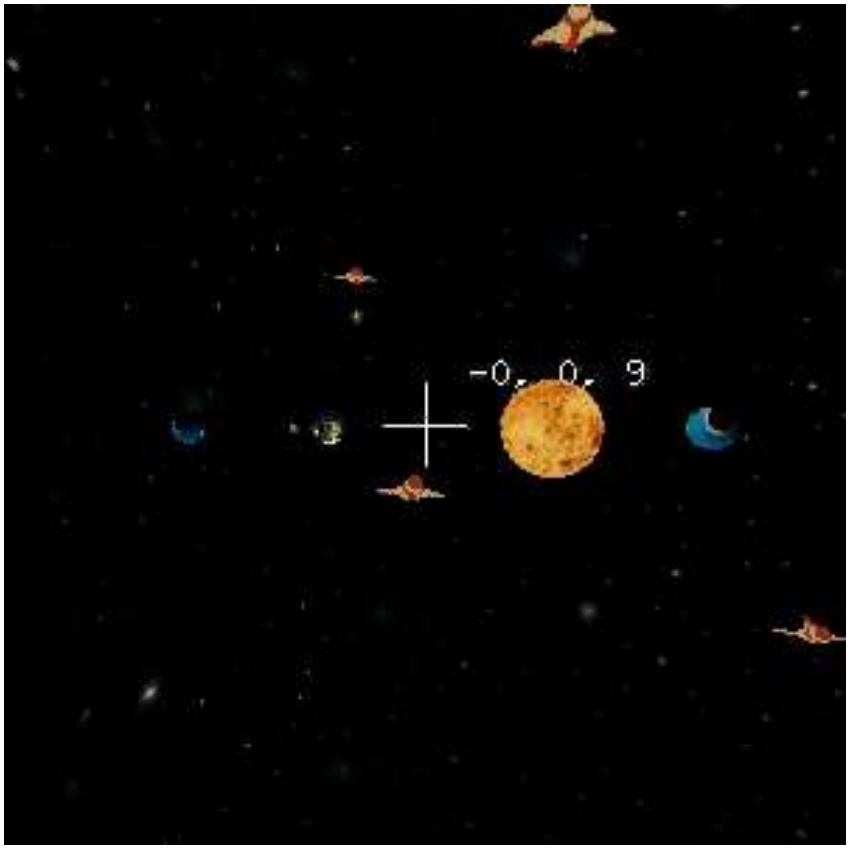
Probléma, ha az objektum gyors

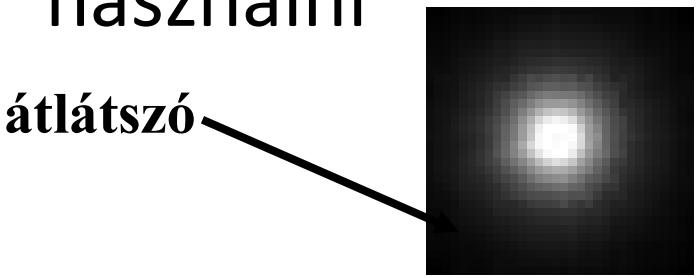



```
dist = obj1.pos - obj2.pos
min = obj1.BoundingRadius() + obj2.BoundingRadius()
if (dist.Length() < min) Collision!
```

Foton torpedó

- Nagyon komplex geometria
- Hasonló kinézet minden irányból
- Könnyebb a képét használni

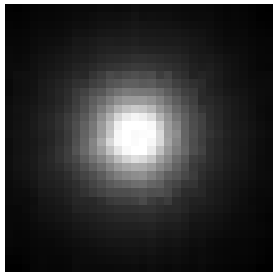


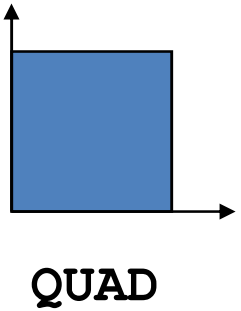


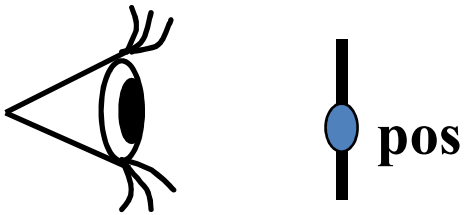
- Ütközésdetektálás = gyors mozgás

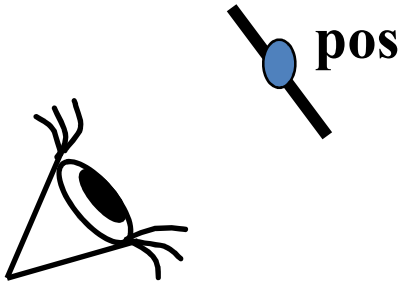
Billboard

Egyetlen félig átlátszó textúra egy téglalapon

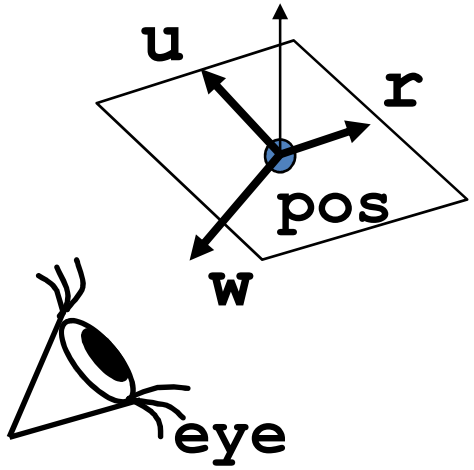








```
void Bullet :: Draw(RenderState state) {
   Vector w = eye - pos;
   Vector r = w % Vector(0, 1, 0);
   Vector u = r % w;
   r = r.normalize() * size;
   u = u.normalize() * size;

   glEnable(GL_BLEND);         // átlátszóság
   glBlendFunc(GL_SRC_ALPHA, GL_ONE); // hozzáadás

   state.M = mat4(r.x,   r.y,   r.z,   0,
                  u.x,   u.y,   u.z,   0,
                  0,     0,     1,     0,
                  pos.x, pos.y, pos.z, 1);

   shader->Bind(state);
   geometry->Draw();

   glDisable(GL_BLEND);
}
```

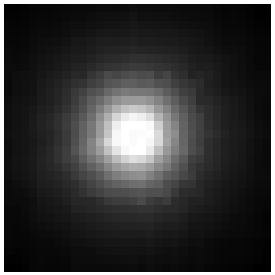


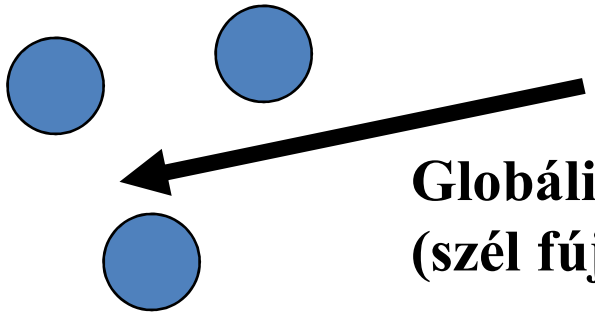
Robbanás

- Nagyon komplex geometria
- Hasonló kinézet minden irányból
- Plakátgyűjtemény

- Részecske rendszer

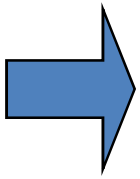
Részecske rendszerek





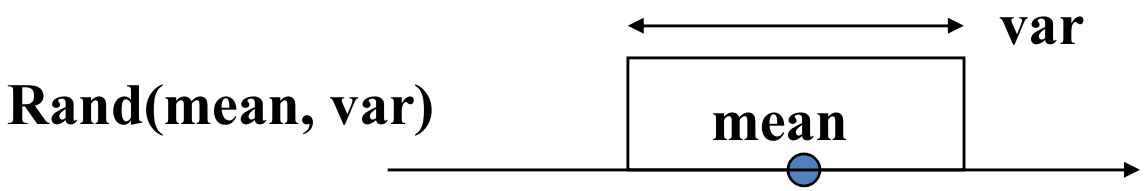
Globális erőtér
(szél fújja a füstöt)

Véletlen Kezdeti értékek



pos:	**pos += velocity * dt**
velocity:	**velocity += acceleration * dt**
acceleration:	**acceleration = force / weight**
lifetime	
age:	**age += dt; if (age > lifetime) Kill();**
size, dsize:	**size += dsize * dt;**
weight, dweight:	**weight += dweight * dt**
color, dcolor:	**color += dcolor * dt**

Robbanás paraméterei




```
pos = center;                          // kezdetben fókuszált
lifetime = Rand(2, 1);

size = 0.001;                                  // kezdetben kicsi
dsize = Rand(0.5, 0.25) / lifetime;

velocity = Vector(Rand(0,0.4),Rand(0,0.4),Rand(0,0.4));
acceleration = Vector(Rand(0,1),Rand(0,1),Rand(0,1));

                       // Planck törvény: sárga átlátszatlanból vörös átlátszóba
color = Color(1, Rand(0.5, 0.25) 0, 1 );
dcolor = Color(0, -0.25, 0, -1) / lifetime;
```

Avatár

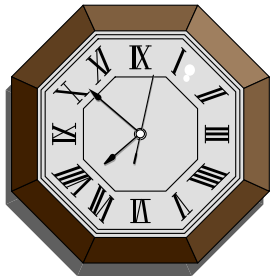
- A viselkedését a klaviatúra vezérli:
  - ProcessInput
- A helye és iránya viszi a kamerát
  - SetCameraTransform
- Olyan mint egy űrhajó, de nem rajzoljuk
  - Control: gravitáció, lövedék ütközés

Klaviatúra kezelés

KeyboardFunc
KeyboardUpFunc
SpecialKeysFunc
SpecialKeysUpFunc

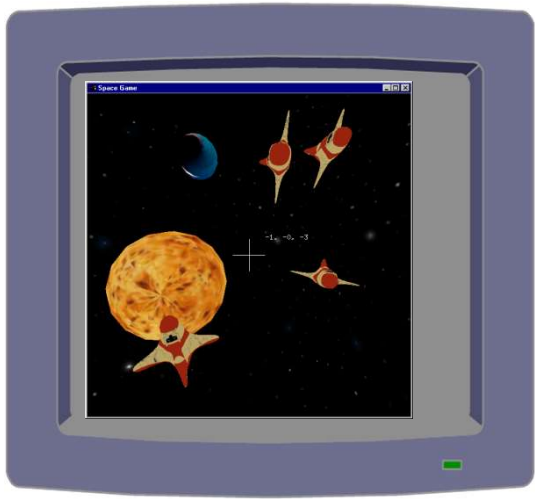
keys

```
bool keys[256];      // is pressed?
void onKeyboard(unsigned char key,
                int pX, int pY) {
    keys[key] = true;
}
void onKeyboardUp(unsigned char key,
                  int pX, int pY) {
    keys[key] = false;
}
```

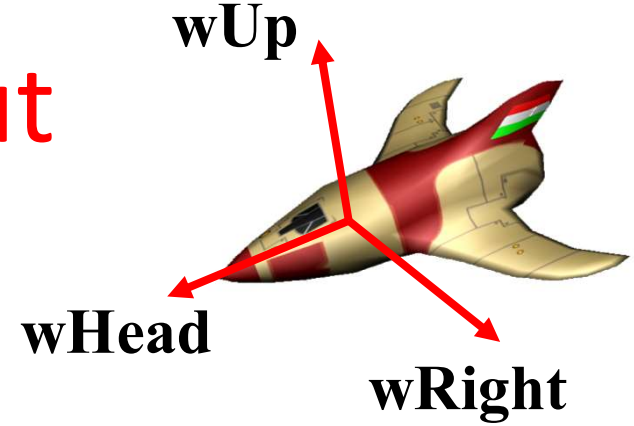


IdleFunc:
GameLoop

virtual
world



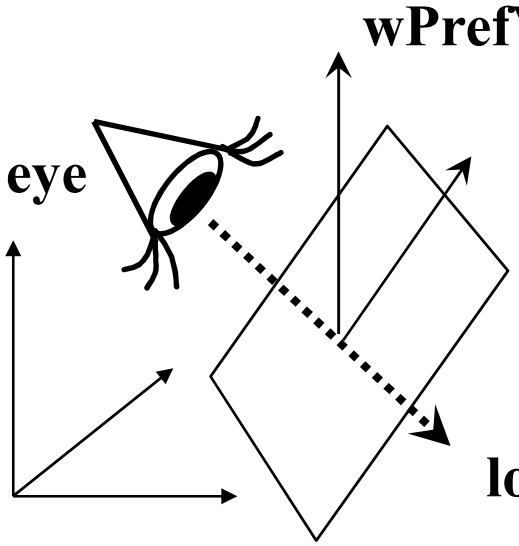
Avatar :: ProcessInput




```
Avatar :: ProcessInput() {
  if ( keys[' '] ) // fire!
    objects.push_back(new Bullet(pos, velocity));

      // Kormányzás: az avatár koordinátarendszerében!
  vec3 wHead = velocity.normalize( );
  vec3 wRight = cross(wPrefVup, wHead).normalize();
  vec3 wUp = cross(wHead, wRight);

  if (keys[KEY_UP])    force -= wUp;
  if (keys[KEY_DOWN])  force += wUp;
  if (keys[KEY_LEFT])  force -= wRight;
  if (keys[KEY_RIGHT]) force += wRight;
}
```

Avatar::SetCameraTransform

```
Avatar :: SetCameraTransform(RenderState& state) {
    Camera camera(pos, pos + velocity, wPrefVup,
                  fov, asp, fp, bp);
    state.V() = camera.V();
    state.P() = camera.P();
}
```



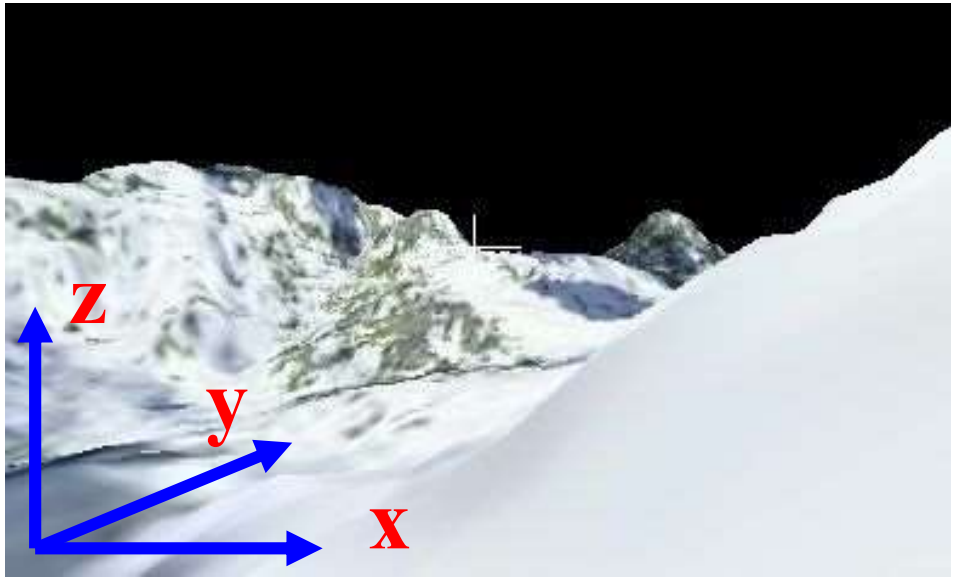
wPrefVup = [0, 1, 0] vagy a
gyorsulásból és a korábbi wPrefVup átlagából

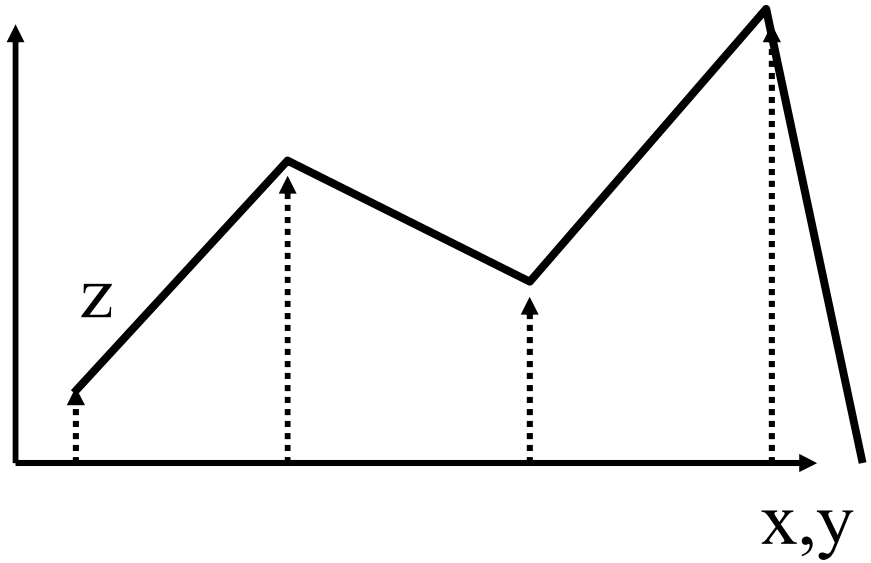
Egy földi lövöldözős játék

Terepek

- Komplex geometria
  - magasságmező
- Bonyolult textúra
- Nem gondolkodik
- Nem mozog
- Ütközés detektálás kell
- Megemeli az objektumokat

Terep geometria

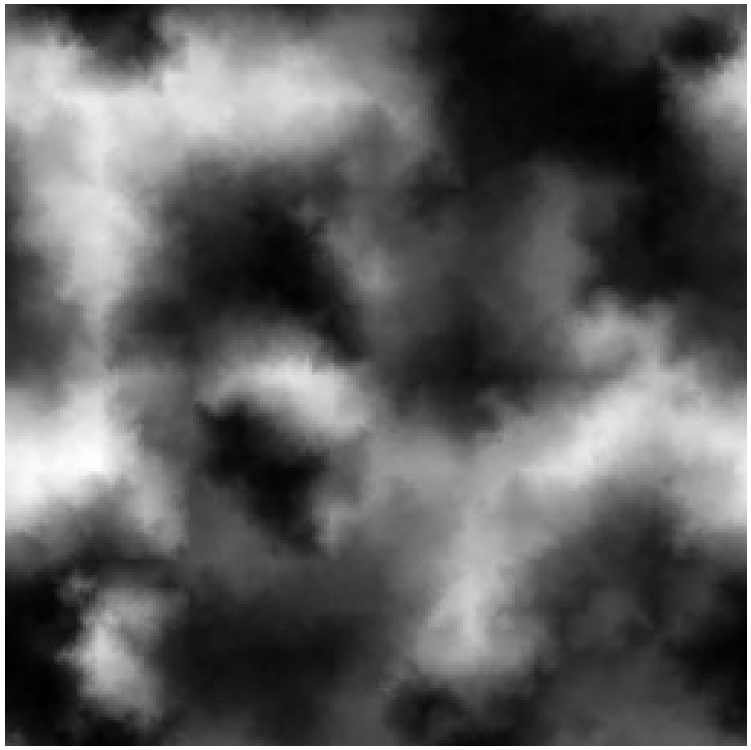




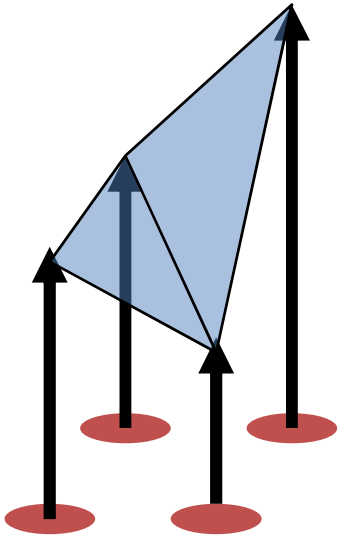
z = height(x,y)

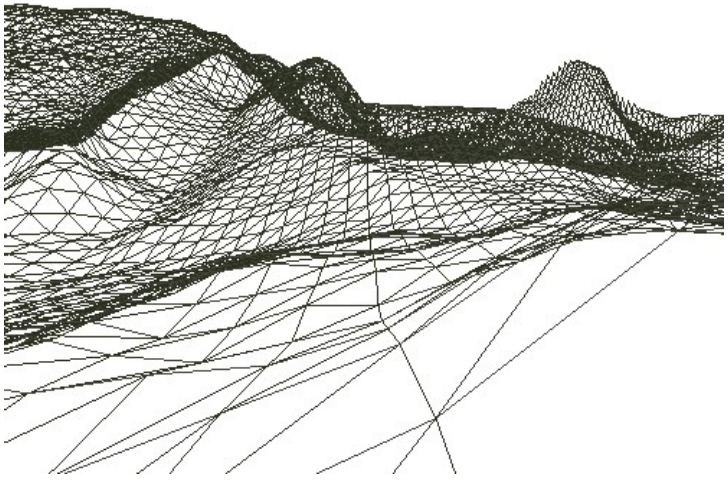
Magasságmező:
Diszkrét minták +
Lineáris interpoláció

Diszkrét minták



Magasság mező



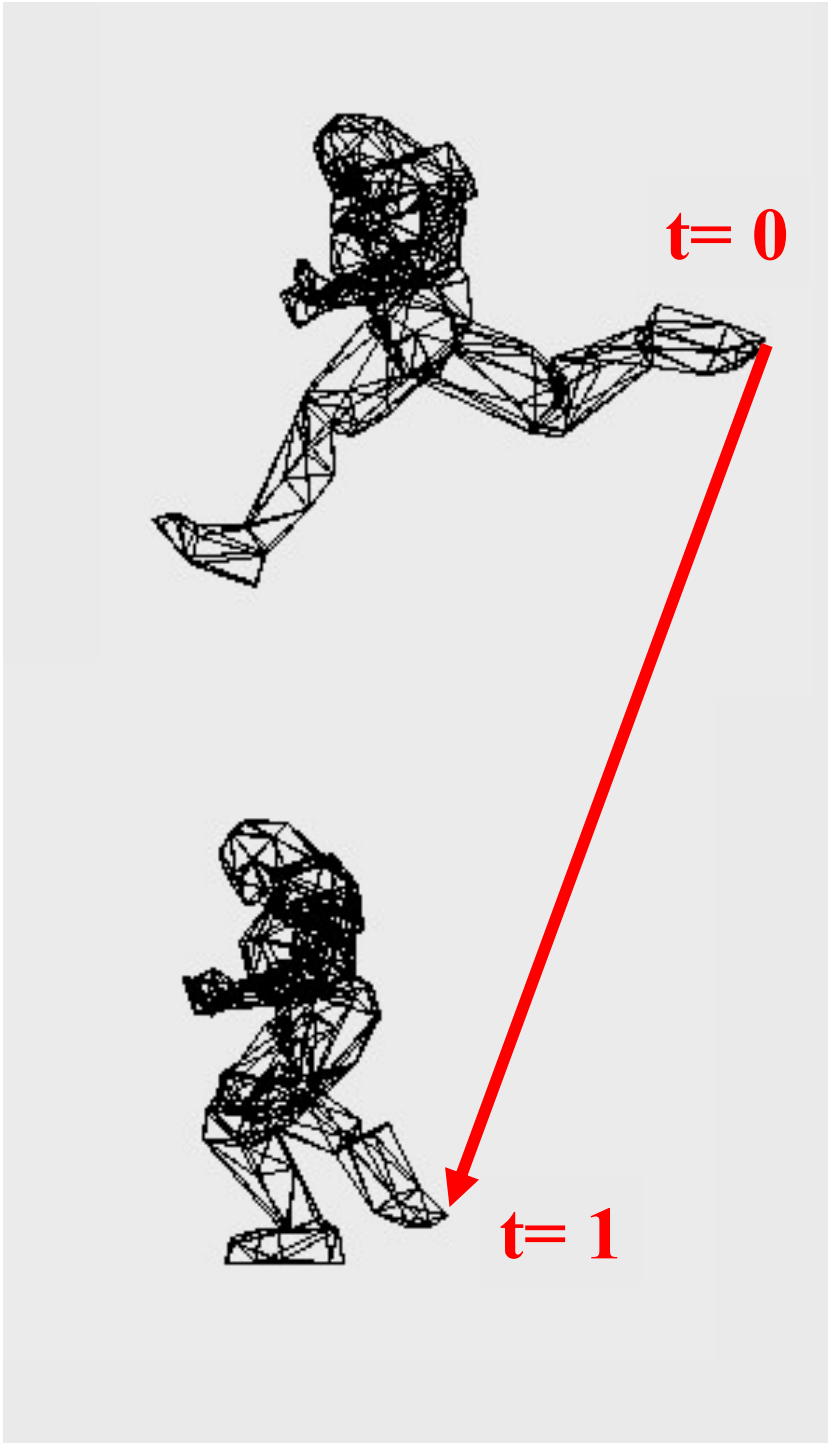


Háromszög háló

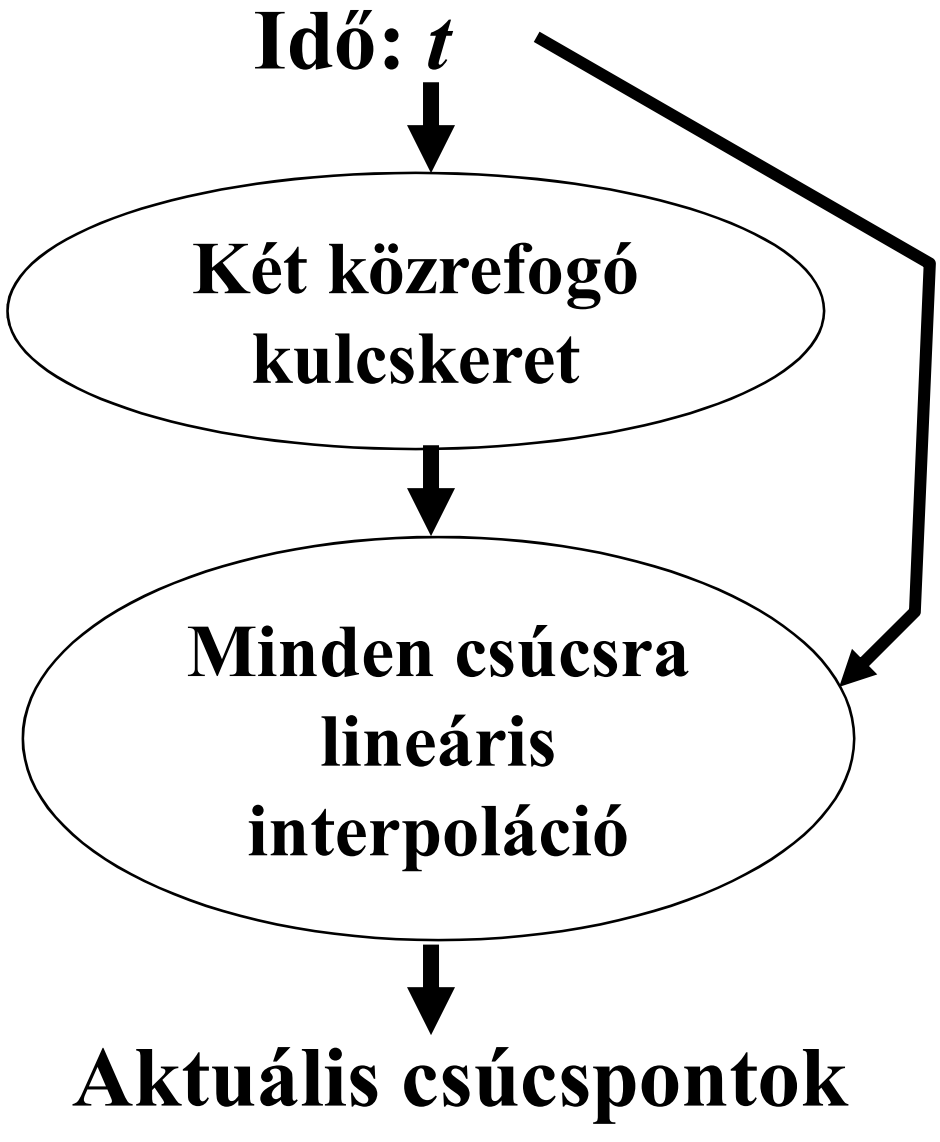
Ellenség

- Animált geometria
  - Kulcskeretekkel (clip-enként)
    - Áll, fut, támad, meghal
  - poligonháló deformáció
- Textúrák (animált)
- AI
- Ütközés detektálás

Kulcskeret animáció: futás



Mesh morphing:



Futás poligonháló deformációval

+ pozíció animáció:

position += velocity * dt

Mozgás definíció

- Clip-ek definíciója kulcskeretekkel
- Összes clip összes kulcskeretek fájlban: MD2, MD3
- Tipikus clip-ek:
  - Run, stand, attack, die, pain, salute, crouch, wave, point, taunt, etc.

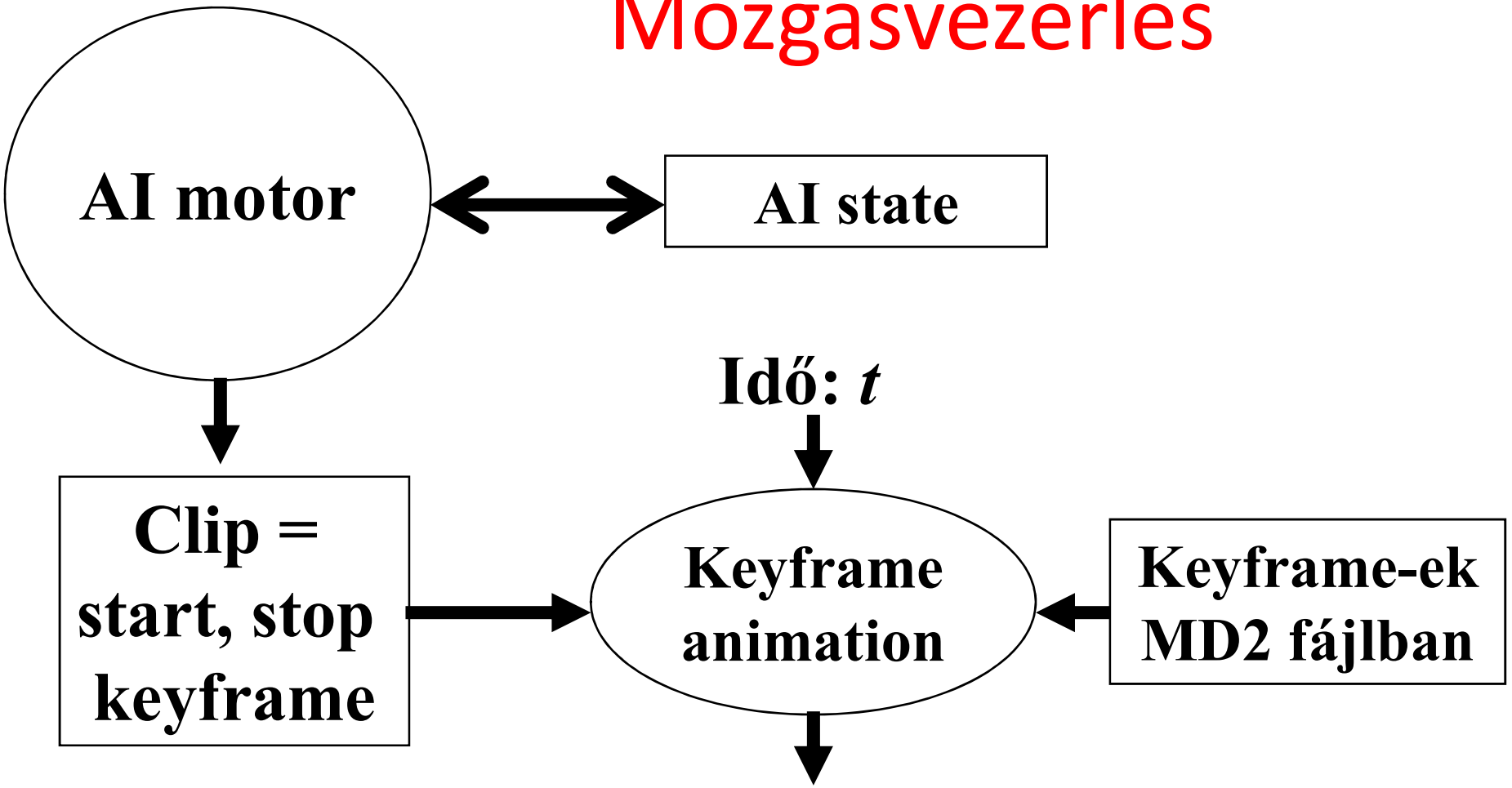
Clip-ek

All
40 kulcskeret

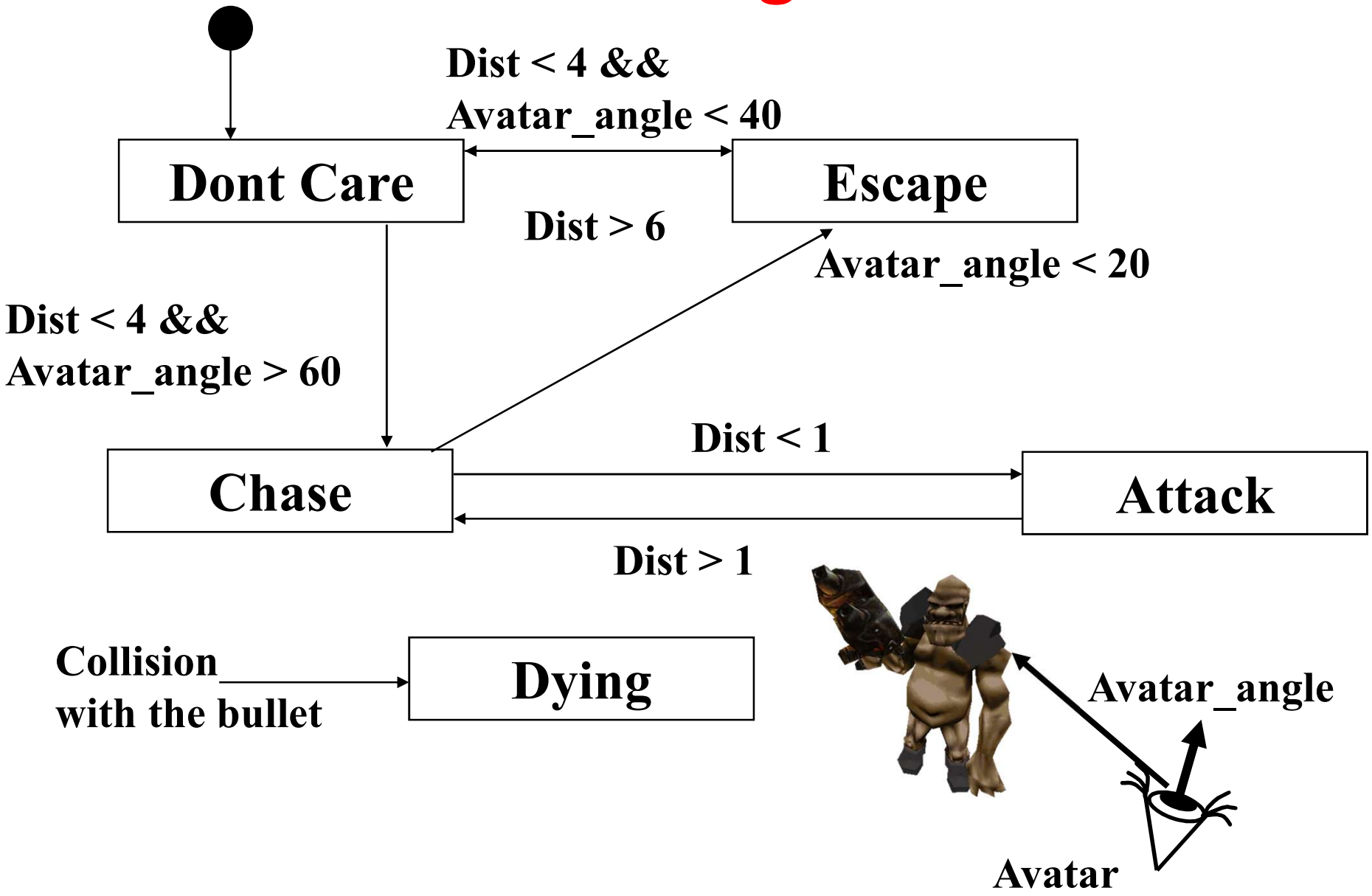
Fut
5 kulcskeret

Szalutál
11 kulcskeret

Mozgásvezérlés



Ellenség AI


Dist < 4 &&
Avatar_angle < 40
Dont Care
Escape
Dist > 6
Avatar_angle < 20
Dist < 4 &&
Avatar_angle > 60
Dist < 1
Chase
Attack
Dist > 1
Collision
with the bullet
Dying
Avatar_angle
Avatar