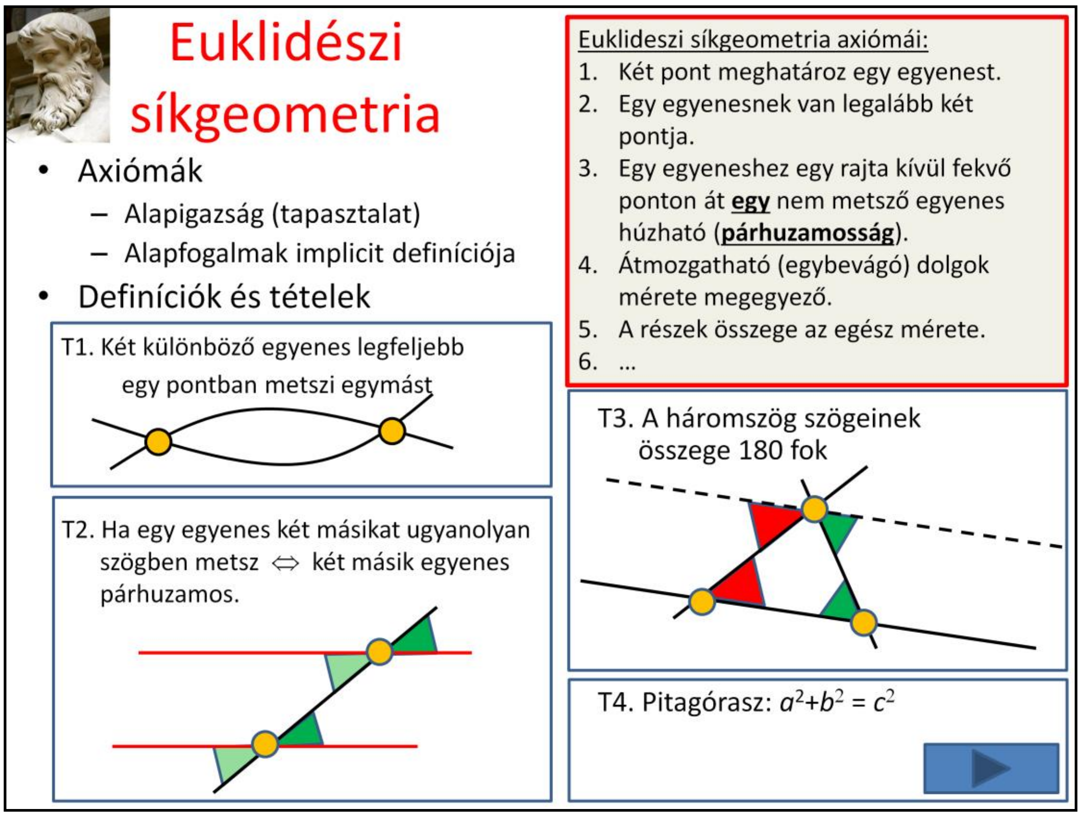
Geometriák és algebrák meselős emlékeztető

Szirmay-Kalos László



Euklidészi síkgeometria

- Axiómák
  - Alapigazság (tapasztalat)
  - Alapfogalmak implicit definíciója
- Definíciók és tételek

Euklideszi síkgeometria axiómái:

1. Két pont meghatároz egy egyenest.
2. Egy egyenesnek van legalább két pontja.
3. Egy egyeneshez egy rajta kívül fekvő ponton át **egy** nem metsző egyenes húzható (**párhuzamosság**).
4. Átmozgatható (egybevágó) dolgok mérete megegyező.
5. A részek összege az egész mérete.
6. …

T1. Két különböző egyenes legfeljebb egy pontban metszi egymást

T2. Ha egy egyenes két másikat ugyanolyan szögben metsz ⇔ két másik egyenes párhuzamos.

T3. A háromszög szögeinek összege 180 fok

T4. Pitagórasz: $a^2+b^2 = c^2$

Computer graphics works with shapes. The field of mathematics that describes shapes is the geometry, so geometry is essential in computer graphics.

Geometry, like other fields of formal science, has **axioms** that are based on experience and cannot be argued but must be accepted as true statements without arguments. From axioms other true statements, called theorems, can be deducted with logic reasoning.

For example, axioms of the **Euclidean geometry** include the postulates shown above.

Axioms have two purposes, on the one hand, they are accepted as true statements. On the other hand, axioms implicitly define **basic concepts** like points, lines etc. because they postulate their properties.

Euclidean geometry is **metric**, i.e. we can talk of the distance between points or separation of lines, called the **angle**, and size is an important concept. Additional axioms introduce the properties of metric quantities (distance and angle).

Having defined the axioms, we can start establishing theorems using logic inference. Such theorems will constitute the geometry. For example, theorems are:

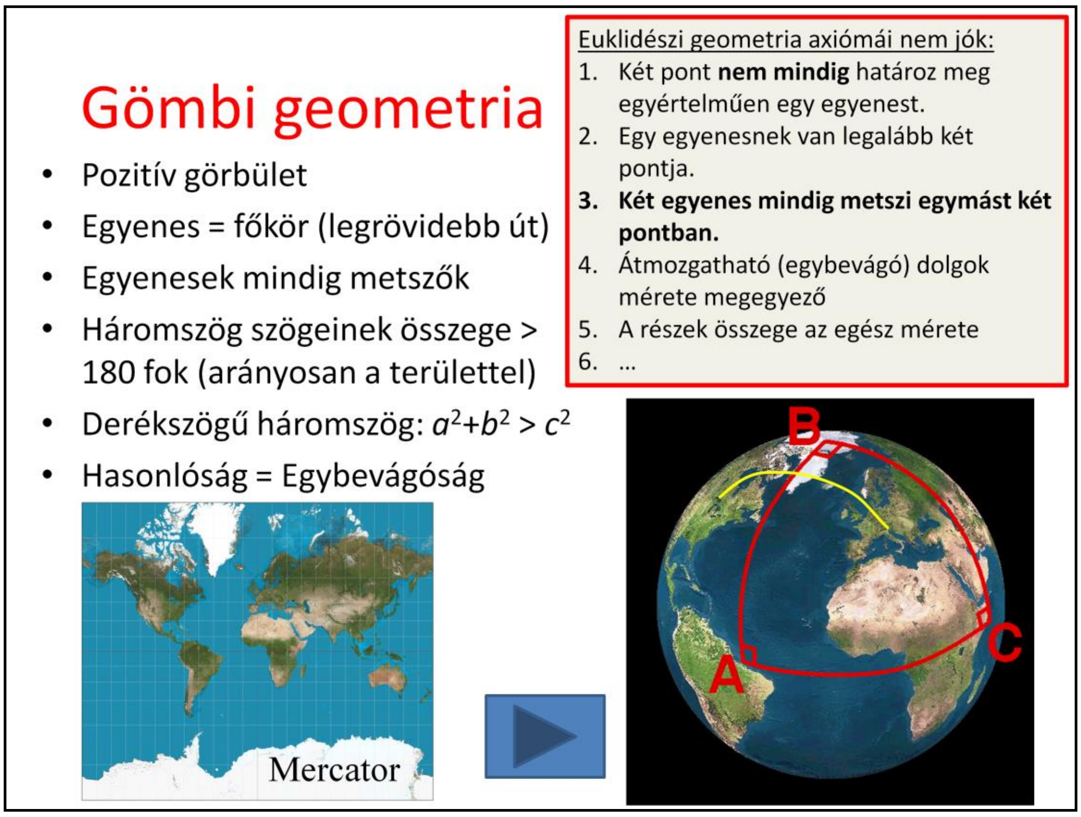
T1: Two different lines may intersect each other at most one point.

T2: Two lines are parallel if and only if the angles in which a third line intersects them are equal.

T3: The sum of angles of a triangle is the half angle, i.e. 180 degrees.

T4: Theorem of Pythagoras.

Prove them using axioms and already proven theorem!



Gömbi geometria

- Pozitív görbület
- Egyenes = főkör (legrövidebb út)
- Egyenesek mindig metszők
- Háromszög szögeinek összege > 180 fok (arányosan a területtel)
- Derékszögű háromszög: $a^2+b^2 > c^2$
- Hasonlóság = Egybevágóság

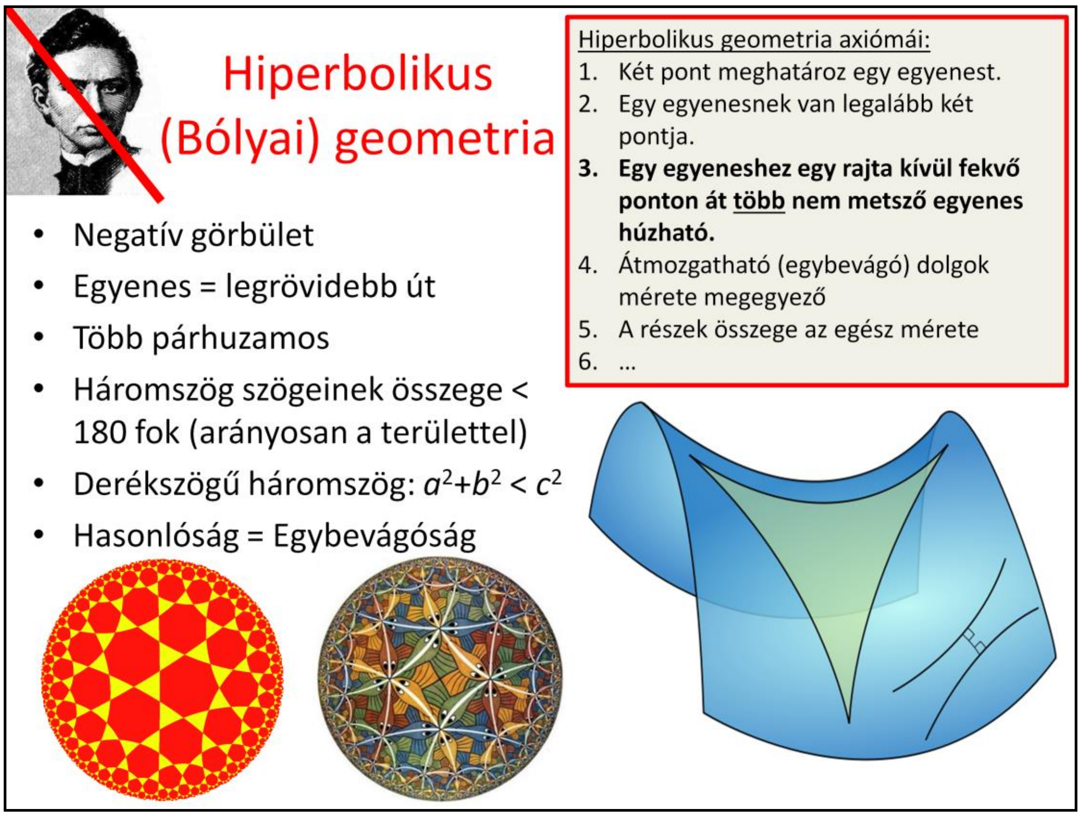
Euklidészi geometria axiómái nem jók:
1. Két pont **nem mindig** határoz meg egyértelműen egy egyenest.
2. Egy egyenesnek van legalább két pontja.
3. **Két egyenes mindig metszi egymást két pontban.**
4. Átmozgatható (egybevágó) dolgok mérete megegyező
5. A részek összege az egész mérete
6. …

The axioms of the Euclidean geometry are based on experience gathered on walking on a flat terrain or not too large distances.

If Euclid had travelled through continents of the spherical Earth, he would have different experience and would have established different axioms. The line on a sphere, using the concept of the shortest path between two points, is the main circle, which is the intersection of the sphere and the plane defined by the two points and the center of the sphere. Airplanes fly along these spherical lines.

The Euclidean axioms are invalid for spherical lines and points. For example, here lines always intersect in two points. Consequently, the theorems of Euclidean geometry are not true. For example, **the sum of the angles of a triangle is always larger than 180 degrees.**

Maps are Euclidean unfolding of the non-Euclidean (spherical) plane. As the curvature of the sphere is not zero, a map must distort distances and/or angles. There are different options, but all of them distort somehow. Mercator’s map, for example, preserves angles but apply drastic distance distortions.



Hiperbolikus (Bólyai) geometria

- Negatív görbület
- Egyenes = legrövidebb út
- Több párhuzamos
- Háromszög szögeinek összege < 180 fok (arányosan a területtel)
- Derékszögű háromszög: $a^2+b^2 < c^2$
- Hasonlóság = Egybevágóság

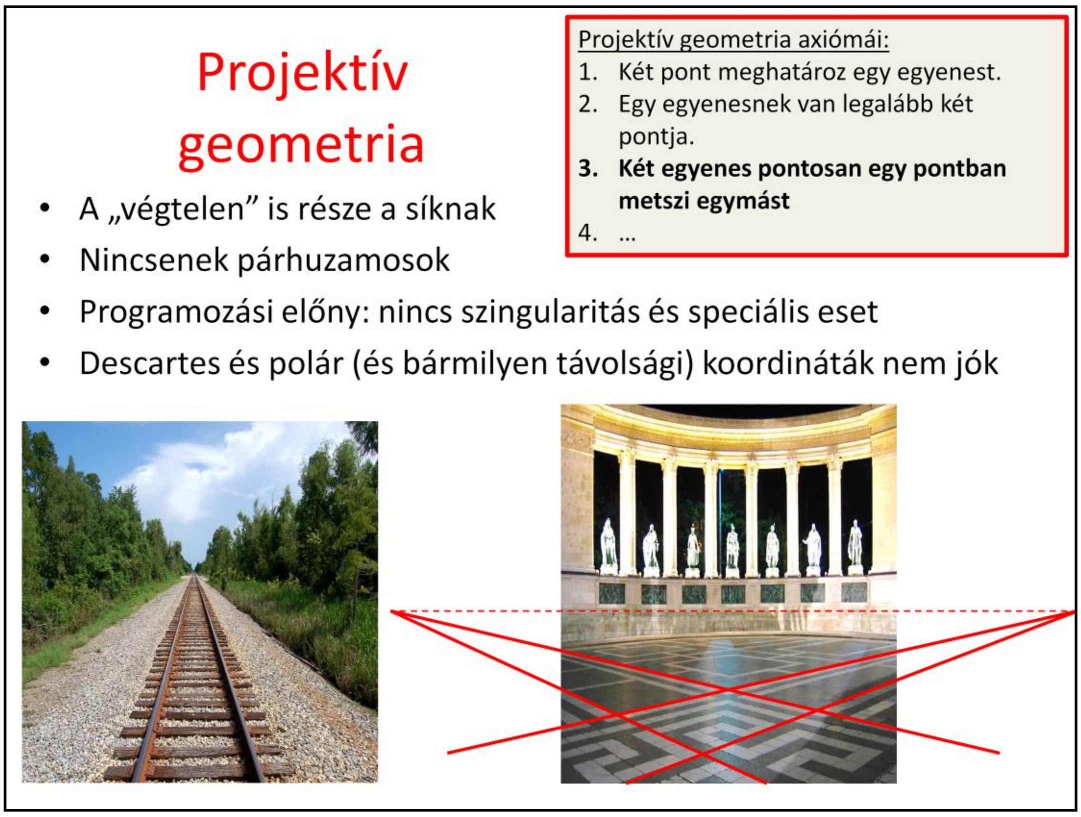
Hiperbolikus geometria axiómái:

1. Két pont meghatároz egy egyenest.
2. Egy egyenesnek van legalább két pontja.
3. **Egy egyeneshez egy rajta kívül fekvő ponton át több nem metsző egyenes húzható.**
4. Átmozgatható (egybevágó) dolgok mérete megegyező
5. A részek összege az egész mérete
6. …

Spaces of negative curvature need another geometry called **hyperbolic geometry**. We should change just a single word in the axioms: there can be more than one parallel line crossing a given point. This small modification may invalidate most of the theorems of Euclidean geometry and lead into a “new world”.

In **hyperbolic geometry the sum of angles of a triangle is less than 180 degrees, proportionally to the size.**

How can we figure it out whether our universe has positive, negative or zero curvature? This is an important question since it determines whether the universe will expand without limits or will start shrinking sooner or later.



Projektív geometria

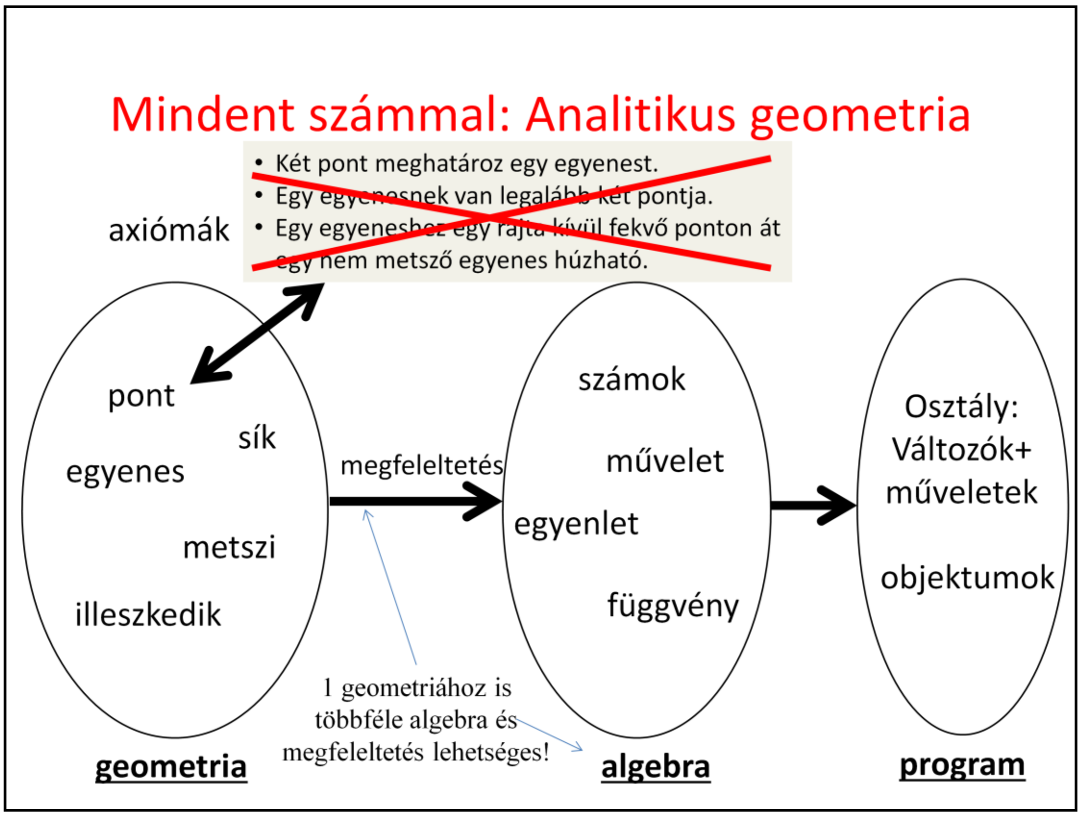
- A „végtelen” is része a síknak
- Nincsenek párhuzamosok
- Programozási előny: nincs szingularitás és speciális eset
- Descartes és polár (és bármilyen távolsági) koordináták nem jók

Projektív geometria axiómái:

1. Két pont meghatároz egy egyenest.
2. Egy egyenesnek van legalább két pontja.
3. **Két egyenes pontosan egy pontban metszi egymást**
4. …

In Euclidean geometry parallel lines do not intersect, that is, a point at infinity (where parallel line would meet) is not part of the Euclidean plane. However, we can see the intersection of parallel lines, so a geometry where infinity is also included makes sense.

If we define axioms differently, **we can add points at infinity to the plane** making all lines, even parallel lines, intersecting. Clearly, this is a different geometry with different axioms and theorems, which is called the **projective geometry**. Projective geometry is not metric since distance cannot be defined in it. The reason is that the distance from points at infinity is infinite, but infinite is not a number. As a result, we cannot use coordinate systems that are based on the concept of distance, e.g. Cartesian coordinate systems are useless here. We should find another algebraic basis.



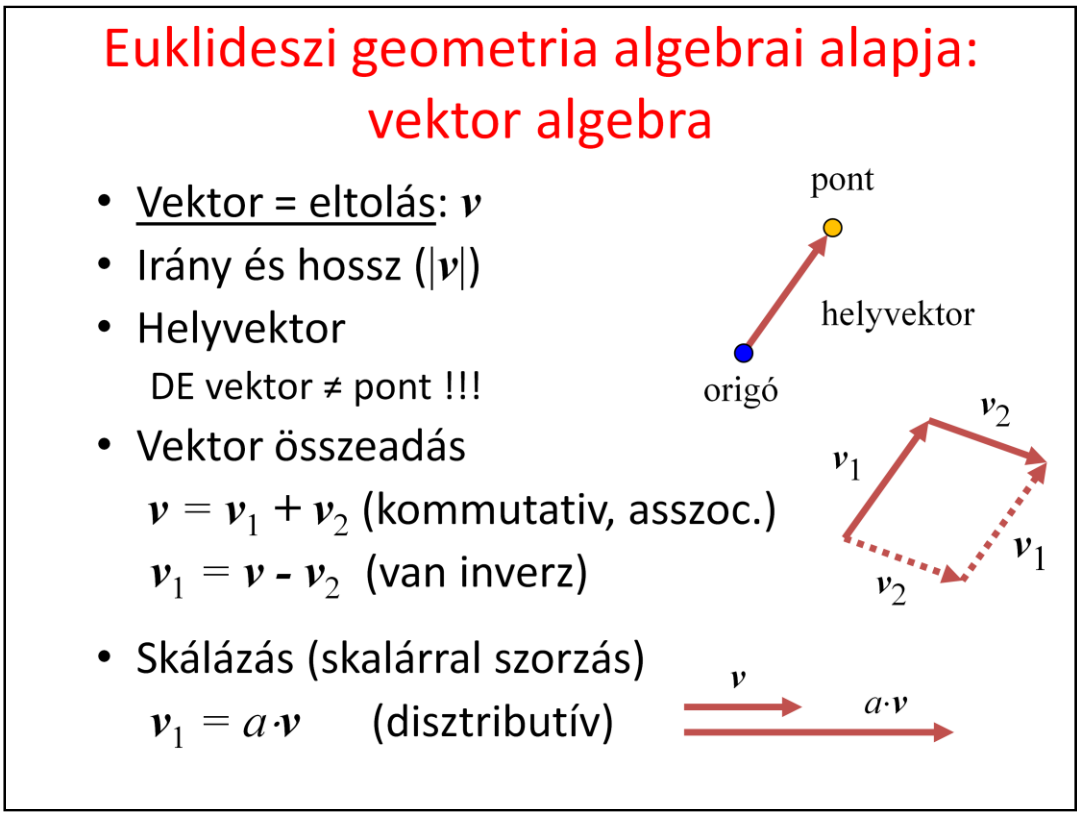
Mindent számmal: Analitikus geometria

axiómák

- ~~Két pont meghatároz egy egyenest.~~
- ~~Egy egyenesnek van legalább két pontja.~~
- ~~Egy egyeneshez egy rajta kívül fekvő ponton át egy nem metsző egyenes húzható.~~

In computer graphics, we should also take into account that a computer is programmed, which cannot do anything else but making calculations with numbers. A computer is definitely not able to understand abstract concepts like point, line etc. So for the application of a computer, geometric concepts must be translated to numbers, calculation and algebra.

A geometry based on algebra, equations and numbers is called **analytic geometry** or **coordinate geometry**. To establish an analytic version of a geometry, we have to find correspondences between geometric concepts and concepts of algebra in a way that axioms of the geometry will not contradict to the concepts of algebra. If it is done, we can forget the original axioms and work only with numbers and equations.



Euklideszi geometria algebrai alapja: vektor algebra

- Vektor = eltolás: $\boldsymbol{v}$
- Irány és hossz ($|\boldsymbol{v}|$)
- Helyvektor

  DE vektor ≠ pont !!!
- Vektor összeadás

  $\boldsymbol{v} = \boldsymbol{v}_1 + \boldsymbol{v}_2$ (kommutativ, asszoc.)

  $\boldsymbol{v}_1 = \boldsymbol{v} - \boldsymbol{v}_2$ (van inverz)
- Skálázás (skalárral szorzás)

  $\boldsymbol{v}_1 = a \cdot \boldsymbol{v}$ (disztributív)

pont
helyvektor
origó
v_2
v_1
v_1
v_2
v
$a \cdot v$

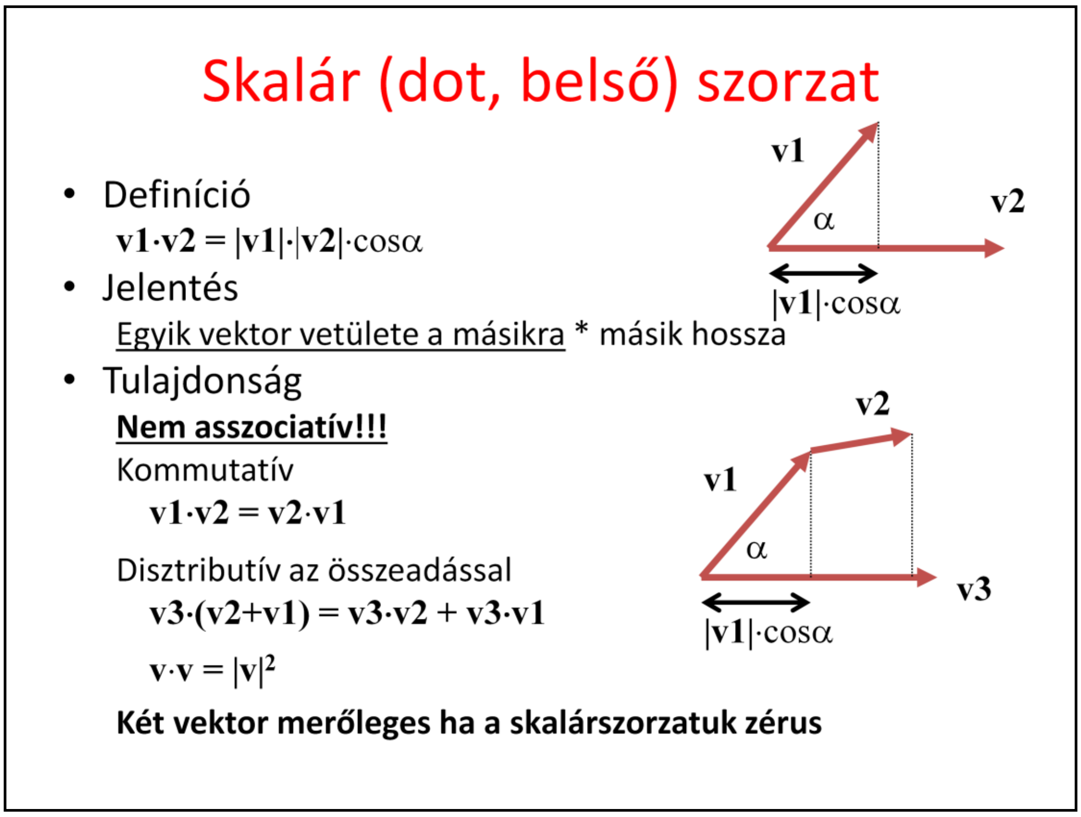
In addition to combining points, we can also **translate** them. By definition a translation is a **vector**, which has **direction** and **length**. The length is denoted by the absolute value of the vector. If we select a special reference point, called the **origin**, then every point has a unique vector that translates the origin to here, or from the other point of view, every vector unambiguously defines a point that is reached if the origin is translated by this vector. Such vectors are called **position vectors**. The fact that there is a one-to-one correspondence between points and position vectors does not mean that points and vectors would be identical objects (wife and husband are also strongly related and unambiguously identify each other, but are still different objects with specific operations).

Concerning vector operations, we can talk of **addition** that means the execution of the two translations one after the other. The resulting translation is independent of the order, so vector addition is **commutative** (parallelogram rule). If we have more than two vectors, parentheses can rearranged so it is also **associative.** Vector addition has an inverse, because we can ask which vector completes the translation of v2 to get a resulting translation v.

Vectors can be **multiplied by a scalar**, which scales the length but does not modify the direction. Scaling is **distributive**, i.e. scaling a sum of two vectors results in the same vector as adding up the two scaled versions.

We have to emphasize that the nice properties of commutativity, associativity, and distributivity are usually not evident and sometimes not even true for vector operations.

Be careful!



Skalár (dot, belső) szorzat

- Definíció

  $\mathbf{v1}\cdot\mathbf{v2} = |\mathbf{v1}|\cdot|\mathbf{v2}|\cdot\cos\alpha$

- Jelentés

  Egyik vektor vetülete a másikra * másik hossza

- Tulajdonság

  Nem asszociatív!!!

  Kommutatív

  $\mathbf{v1}\cdot\mathbf{v2} = \mathbf{v2}\cdot\mathbf{v1}$

  Disztributív az összeadással

  $\mathbf{v3}\cdot(\mathbf{v2}+\mathbf{v1}) = \mathbf{v3}\cdot\mathbf{v2} + \mathbf{v3}\cdot\mathbf{v1}$

  $\mathbf{v}\cdot\mathbf{v} = |\mathbf{v}|^2$

  Két vektor merőleges ha a skalárszorzatuk zérus

Vectors can be multiplied in different ways. The first possibility is the **scalar product** (aka **dot** or inner product) that assigns a scalar to two vectors. By definition, the resulting scalar is equal to the product of the lengths of the two vectors and the cosine of the angle between them.

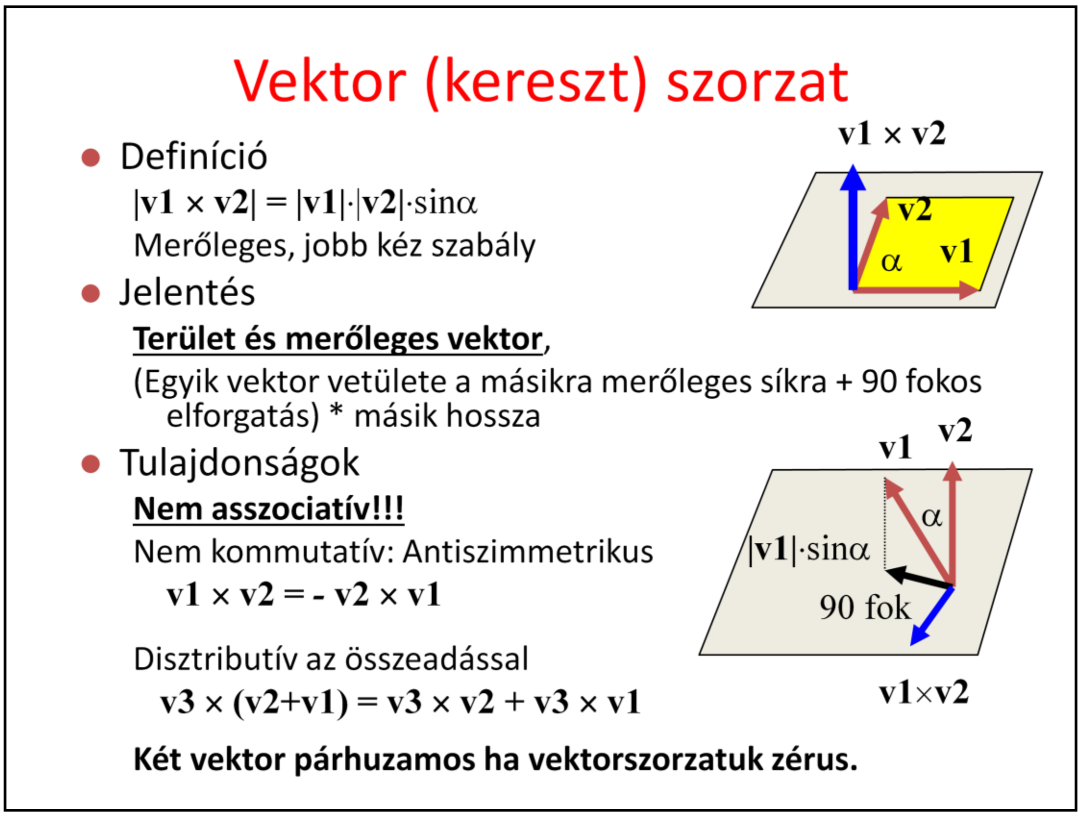
The geometric meaning of scalar product is the length of projection of one vector onto the other, multiplied by the lengths of the others.

Scalar product is **commutative** (symmetric), which is obvious from the definition.

Scalar product is **distributive with the vector addition**, which can be proven by looking at the geometric interpretation. Projection is obviously distributive (the projection of the sum of two vectors is the same as the sum of the two projections.

Scalar product is **NOT associative**!

There is a direct relationship between dot product and the absolute value. The scalar product of a vector with itself is equal to the square of its length according to the definition since cos(0)=1.



Vektor (kereszt) szorzat

- Definíció

  $|\mathbf{v1} \times \mathbf{v2}| = |\mathbf{v1}|\cdot|\mathbf{v2}|\cdot\sin\alpha$

  Merőleges, jobb kéz szabály
- Jelentés

  Terület és merőleges vektor,

  (Egyik vektor vetülete a másikra merőleges síkra + 90 fokos elforgatás) * másik hossza
- Tulajdonságok

  Nem asszociatív!!!

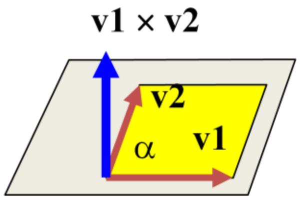
  Nem kommutatív: Antiszimmetrikus

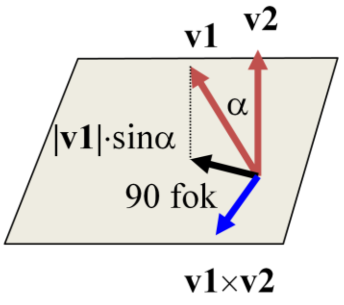
  $\mathbf{v1} \times \mathbf{v2} = - \mathbf{v2} \times \mathbf{v1}$

  Disztributív az összeadással

  $\mathbf{v3} \times (\mathbf{v2}+\mathbf{v1}) = \mathbf{v3} \times \mathbf{v2} + \mathbf{v3} \times \mathbf{v1}$

  Két vektor párhuzamos ha vektorszorzatuk zérus.





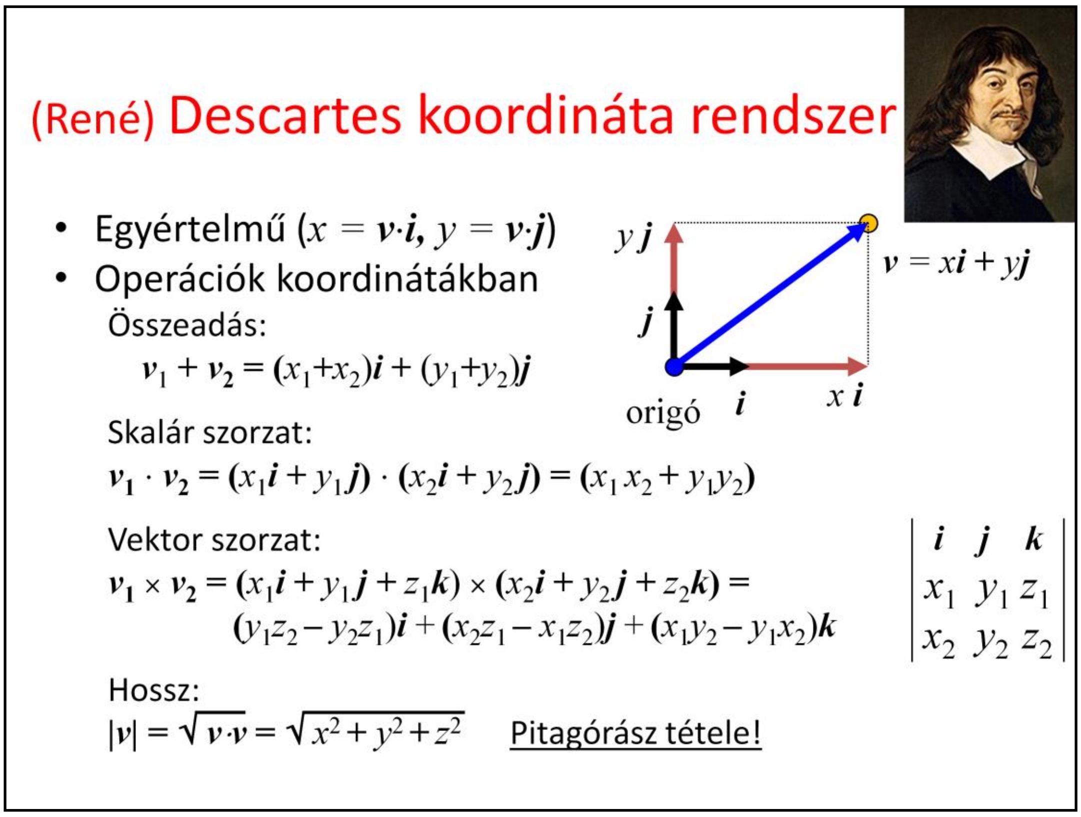
Vectors can be multiplied with the rules of the **vector (aka cross) product** as well. The result is a vector of length equal to the product of the lengths of the two vectors and the sine of their angle. The resulting vector is perpendicular to both operands and points into the direction of the middle finger of our right hand if our thumb points into the direction of the first operand and our index finger into the direction of the second operand (**right hand rule**).

Cross product can be given two different geometric interpretations. The first is a vector meeting the requirements of the right hand rule and of length equal to the area of the parallelogram of edge vectors of the two operands.

The second geometric interpretation is the projection of the second vector onto the plane perpendicular to the first vector, rotating the projection by 90 degrees around the first vector, and finally scaling the result with the length of the first vector.

Cross product is **NOT commutative** but **anti-symmetric** or alternating, which means that exchanging the two operands the result is multiplied by -1.

Cross product is **distributive with the addition**, which can be proven by considering its second geometric interpretation. Projection onto a plane is distributive with addition, so are rotation and scaling. Cross product is **NOT associative**.



(René) Descartes koordináta rendszer

- Egyértelmű ($x = \boldsymbol{v}\cdot\boldsymbol{i}$, $y = \boldsymbol{v}\cdot\boldsymbol{j}$)
- Operációk koordinátákban

Összeadás:

$$\boldsymbol{v}_1 + \boldsymbol{v}_2 = (x_1+x_2)\boldsymbol{i} + (y_1+y_2)\boldsymbol{j}$$

Skalár szorzat:

$$\boldsymbol{v}_1 \cdot \boldsymbol{v}_2 = (x_1\boldsymbol{i} + y_1\boldsymbol{j}) \cdot (x_2\boldsymbol{i} + y_2\boldsymbol{j}) = (x_1 x_2 + y_1 y_2)$$

Vektor szorzat:

$$\boldsymbol{v}_1 \times \boldsymbol{v}_2 = (x_1\boldsymbol{i} + y_1\boldsymbol{j} + z_1\boldsymbol{k}) \times (x_2\boldsymbol{i} + y_2\boldsymbol{j} + z_2\boldsymbol{k}) = (y_1 z_2 - y_2 z_1)\boldsymbol{i} + (x_2 z_1 - x_1 z_2)\boldsymbol{j} + (x_1 y_2 - y_1 x_2)\boldsymbol{k}$$

$$\begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}$$

Hossz:

$$|\boldsymbol{v}| = \sqrt{\boldsymbol{v}\cdot\boldsymbol{v}} = \sqrt{x^2 + y^2 + z^2}$$

Pitagórász tétele!

Having vectors and operations, we are ready to establish a **Cartesian coordinate system**. Let us select one point of the plane and two unit (length) vectors ***i*** and ***j*** that are perpendicular to each other. A vector has unit length if its scalar product with itself is 1 and two vectors are perpendicular if their scalar product is zero since cos(90)=0 (formally: $\boldsymbol{i}\cdot\boldsymbol{i} = \boldsymbol{j}\cdot\boldsymbol{j} = 1$ and $\boldsymbol{i}\cdot\boldsymbol{j} = 0$).

Now, any position vector ***v*** can be unambiguously given as a linear combination of basis vector ***i*** and ***j***, i.e. in the form $\boldsymbol{v} = x\boldsymbol{i} + y\boldsymbol{j}$, where x and y are scalars, called the **coordinates**. Having ***v***, scalar products determine the appropriate coordinates: $x = \boldsymbol{v}\cdot\boldsymbol{i}$, $y = \boldsymbol{v}\cdot\boldsymbol{j}$. To prove this, let us multiply both sides of $\boldsymbol{v} = x\boldsymbol{i} + y\boldsymbol{j}$ by ***i*** and ***j***.

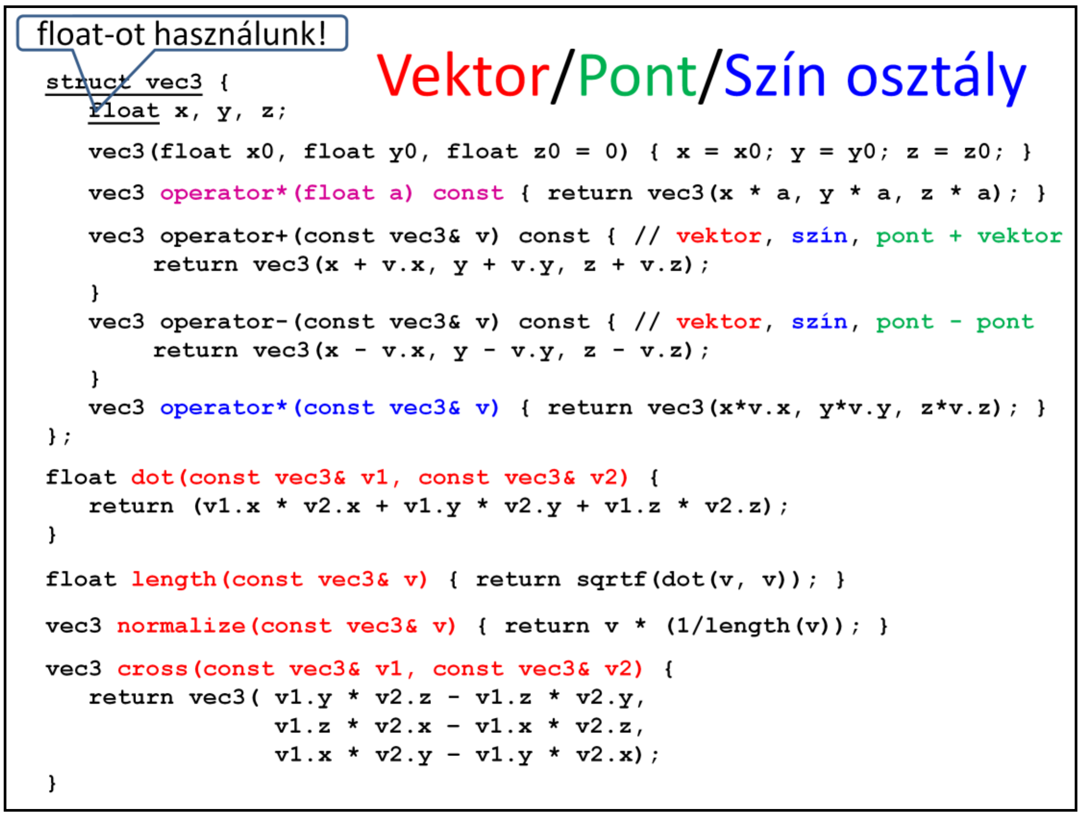
As there is a one-to-one correspondence between vectors and coordinate pairs in 2D (and coordinate triplets in 3D), vectors can be represented by coordinates in all operations.

Based on the associative property of vector addition and on distributive property of multiplying a vector by a scalar with addition, we can prove that coordinates of the sum of two vectors are the sums of the respective coordinates of the two vectors.

Similarly, based on the distributive property of dot product with vector addition, we can prove that the dot product equals to the sum of the products of respective coordinates. Here we also exploit that $\boldsymbol{i}\cdot\boldsymbol{i} = \boldsymbol{j}\cdot\boldsymbol{j} = 1$ and $\boldsymbol{i}\cdot\boldsymbol{j} = 0$.

Finally, based on the distributive property of the cross product with vector addition, we can also express the cross product of two vector with their coordinates. We should also use the cross products of the base vectors $\boldsymbol{i}\times\boldsymbol{i} = 0$, $\boldsymbol{i}\times\boldsymbol{j} = \boldsymbol{k}$, etc. The result can be memorized as a determinant where the first row contains the three basis vectors, the second the coordinates of the first operand, the third the coordinates of the second operand.

The absolute value of a vector is the square root of the scalar product of the vector with itself. Note that we get the Pythagoras theorem for free.



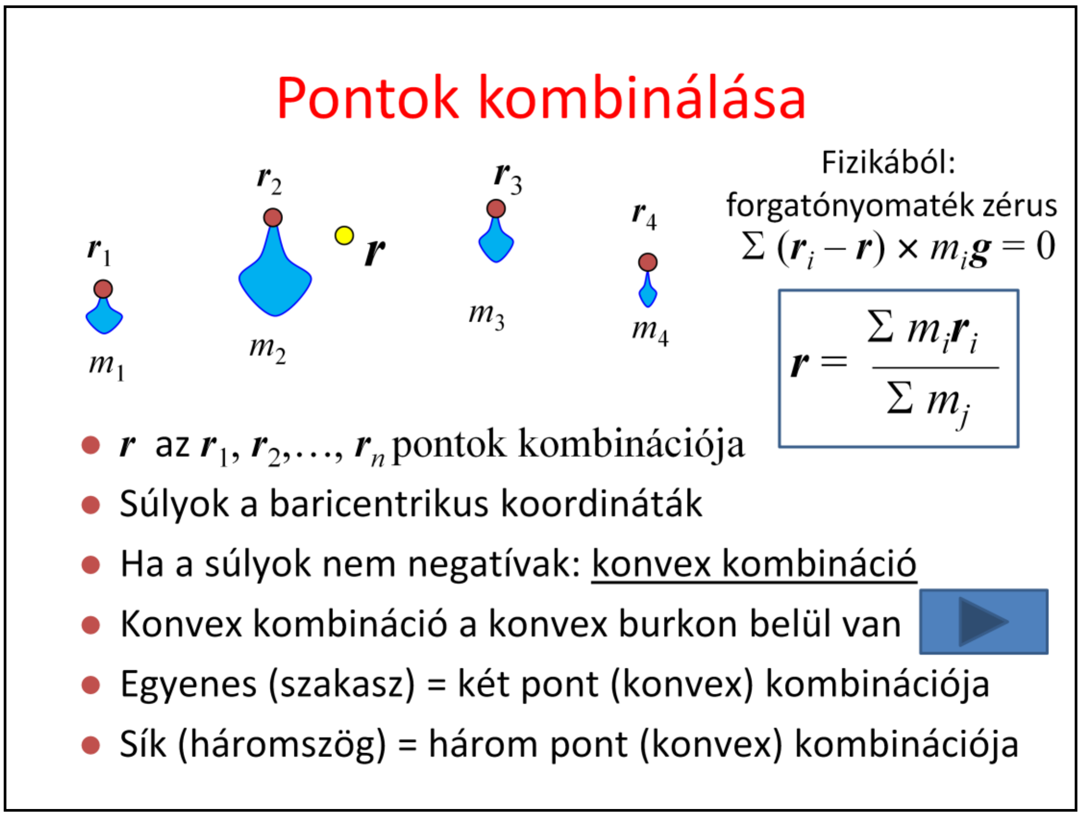
Vektor/Pont/Szín osztály

float-ot használunk!

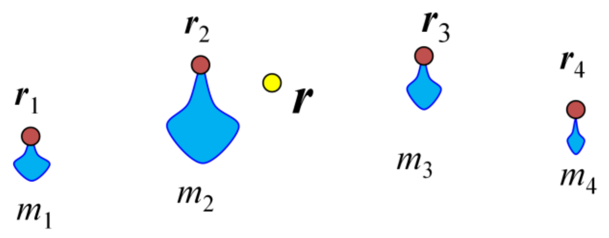
```
struct vec3 {
   float x, y, z;

   vec3(float x0, float y0, float z0 = 0) { x = x0; y = y0; z = z0; }

   vec3 operator*(float a) const { return vec3(x * a, y * a, z * a); }

   vec3 operator+(const vec3& v) const { // vektor, szín, pont + vektor
        return vec3(x + v.x, y + v.y, z + v.z);
   }
   vec3 operator-(const vec3& v) const { // vektor, szín, pont - pont
        return vec3(x - v.x, y - v.y, z - v.z);
   }
   vec3 operator*(const vec3& v) { return vec3(x*v.x, y*v.y, z*v.z); }
};

float dot(const vec3& v1, const vec3& v2) {
   return (v1.x * v2.x + v1.y * v2.y + v1.z * v2.z);
}

float length(const vec3& v) { return sqrtf(dot(v, v)); }

vec3 normalize(const vec3& v) { return v * (1/length(v)); }

vec3 cross(const vec3& v1, const vec3& v2) {
   return vec3( v1.y * v2.z - v1.z * v2.y,
                v1.z * v2.x - v1.x * v2.z,
                v1.x * v2.y - v1.y * v2.x);
}
```

The implementation of the theory discussed so far is a single C++ class representing a 3D point or a vector with three Cartesian coordinates. Using operator overloading, the discussed vector (and point) operations are also.

Note that – similarly to the GLSL language – we use the same type to represent points and vectors. The programmer should be aware whether an object is a point or a vector and execute operations valid for this particular type. Mixing different types in a single class, we can extend the concept to colors as well. A color can be define as additive mixture of red, green, and blue components, so a three-dimensional vector is just a right representation.



Pontok kombinálása



Fizikából:
forgatónyomaték zérus
$$\Sigma\,(\boldsymbol{r}_i - \boldsymbol{r}) \times m_i \boldsymbol{g} = 0$$

$$\boxed{\boldsymbol{r} = \frac{\Sigma\, m_i \boldsymbol{r}_i}{\Sigma\, m_j}}$$

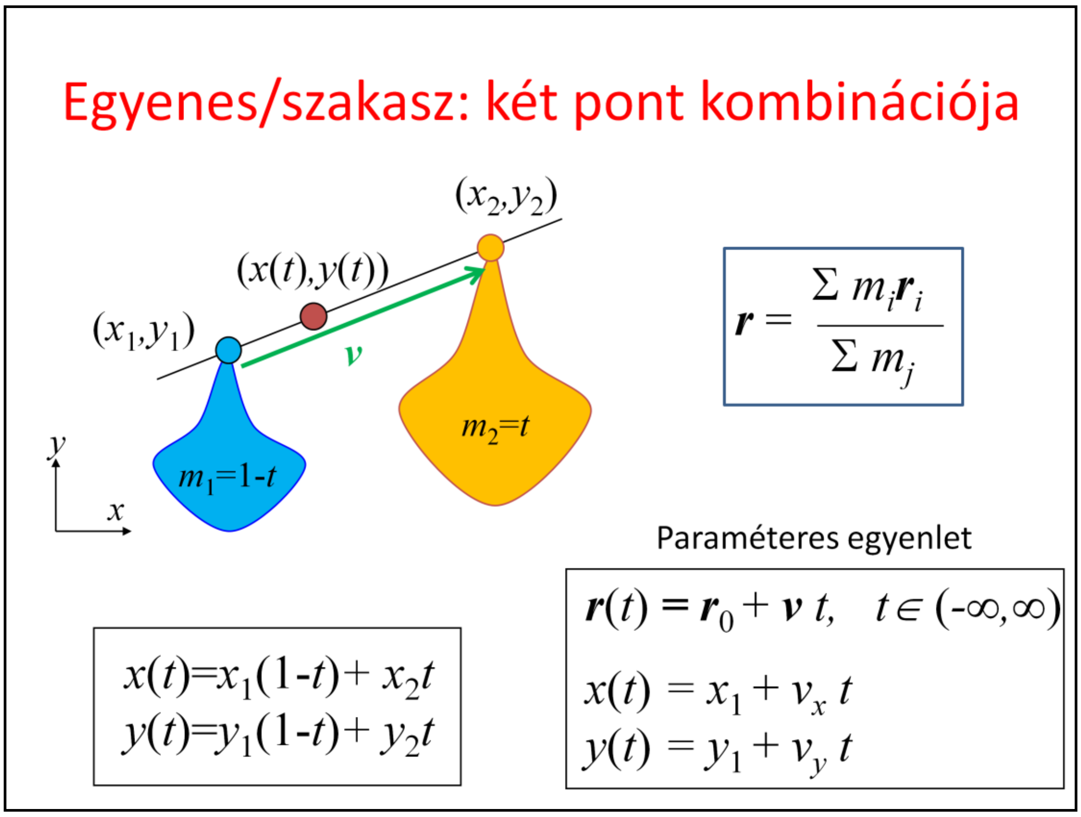
- $\boldsymbol{r}$ az $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n$ pontok kombinációja
- Súlyok a baricentrikus koordináták
- Ha a súlyok nem negatívak: konvex kombináció
- Konvex kombináció a konvex burkon belül van
- Egyenes (szakasz) = két pont (konvex) kombinációja
- Sík (háromszög) = három pont (konvex) kombinációja

Defining a point as the center of mass of a system where masses placed at finite number of reference points is also called the **combination of these points with barycentric coordinates** equal to the weights.

Note that we can do this in real life without mathematics and coordinate systems, center mass exists and is real without mathematics and abstraction.

If all weights are non-negative, which has direct physical meaning, then we talk of **convex combination** since the points that can be defined in this way are in the convex hull of the reference points. By definition, the **convex hull** is the minimal set of points that is convex and includes the original reference points. For example, when presents are wrapped, the wrapping paper is on the convex hull of the presents.

Using the term combination or convex combination, we can define a line as a combination of two points and a line segment as a convex combination of two points. Similarly, the convex combination of three not collinear points is the triangle, the convex combination of four points not being in the same plane is a tetrahedron.



Egyenes/szakasz: két pont kombinációja

$$\boldsymbol{r} = \frac{\sum m_i \boldsymbol{r}_i}{\sum m_j}$$

Paraméteres egyenlet

$$\boldsymbol{r}(t) = \boldsymbol{r}_0 + \boldsymbol{v}\, t, \quad t \in (-\infty,\infty)$$

$$x(t) = x_1 + v_x\, t$$
$$y(t) = y_1 + v_y\, t$$

$$x(t)=x_1(1-t)+ x_2 t$$
$$y(t)=y_1(1-t)+ y_2 t$$

Having points, we can start defining primitives built of infinitely many points. We have two basic operations on points, combination and finding the vector that translates one point to the other.

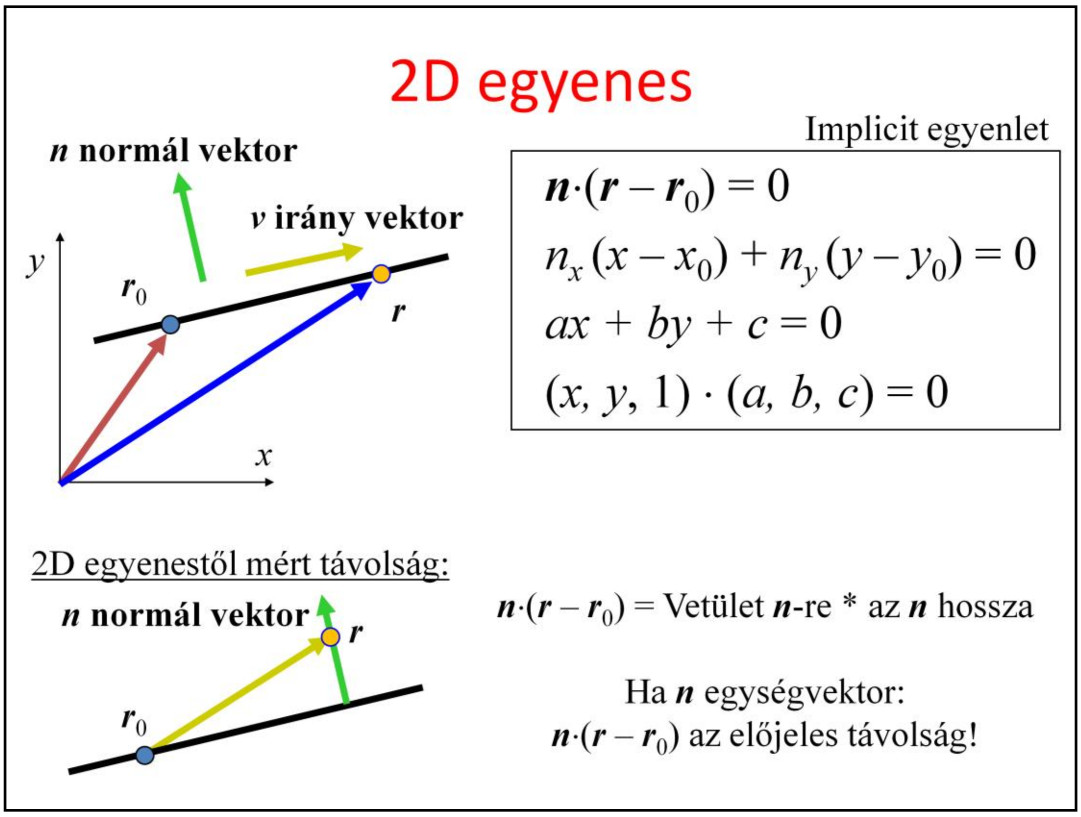
If we have a translation vector, we can ask the distance, impose requirements on orthogonality or parallelism.

Combination uses the center of mass analogy, which assigns the center of mass to a set of points by the given formula. The position vectors of individual points are multiplied by the mass placed there and the sum is divided by the total mass.

Let us select two points that will be combined and, for the sake of simplicity, let us assume that the total mass is 1 (we distribute 1 kg mass in the two points). Distributing unit mass has the advantage that we do not have to divide with the total mass since division by 1 can be saved.

The center of mass will be on a line segment between by the two points. Whether it is closer to the first or to the second point depends on t, so by modifying t in [0,1] we can make the center of mass run on the line segment. So, points of the line segment can be expressed by a function of t. Such equation is called parametric equation because we have a free parameter that controls which point of the primitive is currently selected.

If t can be outside of the unit interval, then a point can also repel the point, thus the center of mass will still be on the line of the two points but outside of the line segment. The equation of the line segment and the line are similar, only the parameter ranges are different. The equation can also be rewritten in another form, where the two points are replaced by one point, called the position vector of the line, and the vector between them, called the direction vector of the line.



2D egyenes

Implicit egyenlet

$$\boldsymbol{n}\cdot(\boldsymbol{r}-\boldsymbol{r}_0)=0$$
$$n_x(x-x_0)+n_y(y-y_0)=0$$
$$ax+by+c=0$$
$$(x,y,1)\cdot(a,b,c)=0$$

2D egyenestől mért távolság:

$\boldsymbol{n}\cdot(\boldsymbol{r}-\boldsymbol{r}_0)$ = Vetület $\boldsymbol{n}$-re * az $\boldsymbol{n}$ hossza

Ha $\boldsymbol{n}$ egységvektor:
$\boldsymbol{n}\cdot(\boldsymbol{r}-\boldsymbol{r}_0)$ az előjeles távolság!

Having points, we can start defining primitives built of infinitely many points.

We have two basic operations on points, combination and finding the vector that translates one point to the other. If we have a translation vector, we can ask the distance, impose requirements on orthogonality or parallelism.

Combination uses the center of mass analogy, which assigns the center of mass to a set of points by the given formula. The position vectors of individual points are multiplied by the mass placed there and the sum is divided by the total mass.

Let us select two points that will be combined and, for the sake of simplicity, let us assume that the total mass is 1 (we distribute 1 kg mass in the two points). Distributing unit mass has the advantage that we do not have to divide with the total mass since division by 1 can be saved.

The other way of establishing the equation of the line is based on orthogonality (or, from another point of view, on distance). The difference vectors of any two points on the line are all parallel, so they are all perpendicular to a given vector, called the normal vector of the line. Let one point be a given point, called the position vector of the line, and the other point represent any point (this is called the running point). Their difference r-r0 is perpendicular to normal vector n if and only if their scalar product is zero. This equation imposes a requirement on running point r. If r satisfies this equation, then the point is on the line, otherwise it is not on the line.

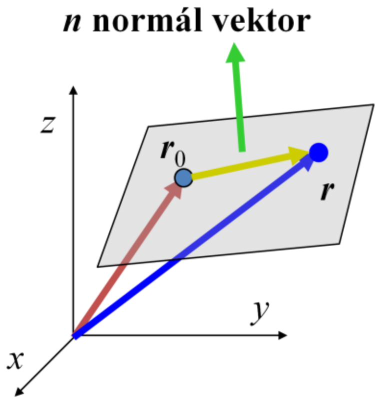
Another interpretation uses the distance. Point r is on the line if its distance from the line is zero. We know from geometry that the distance should be measured in perpendicular direction, which is

$|\boldsymbol{n}\cdot(\boldsymbol{r}-\boldsymbol{r}_0)|$ if $\boldsymbol{n}$ is a unit vector (the difference is projected onto the unit normal vector).

Expressing the line equation with coordinates, we get an implicit linear equation for unknown point coordinates x and y. If a point's x,y coordinates satisfy this equation, the point is on the line.

This implicit equation can also be expressed by the scalar product of two three-element vectors if we use the convention that 2D points have three coordinates where the third coordinate is 1.

Sík

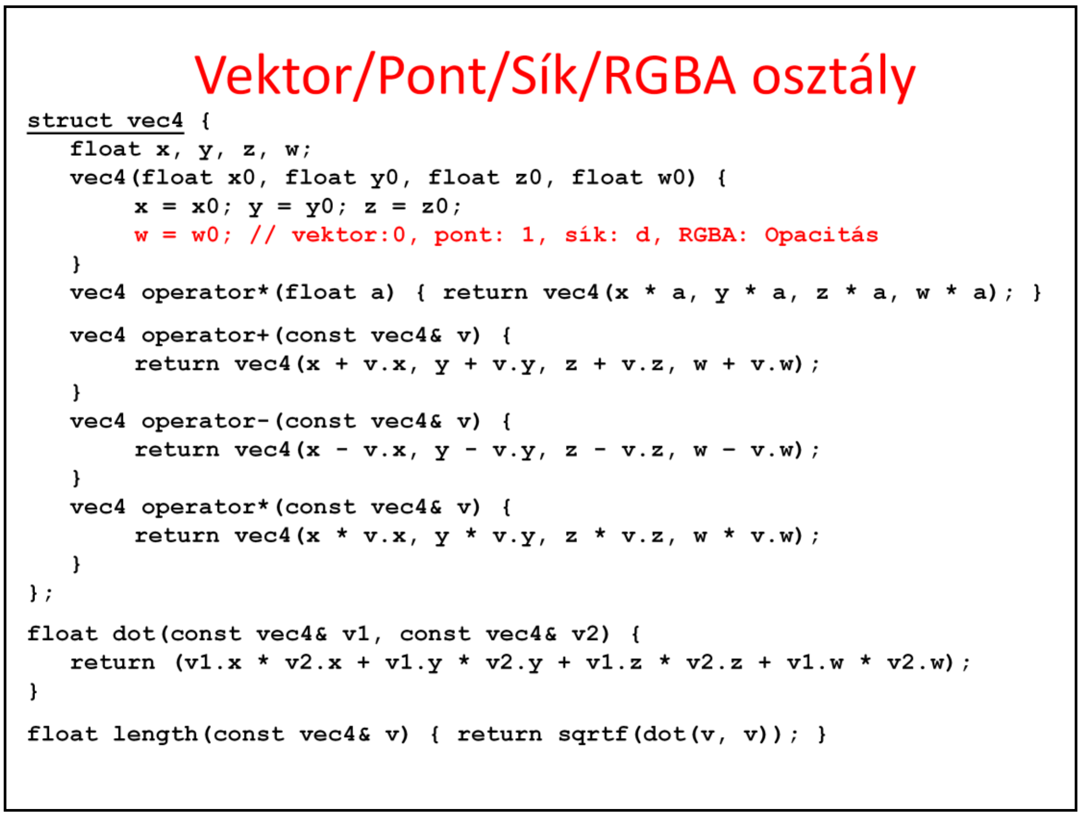


$$\boldsymbol{n}\cdot(\boldsymbol{r}-\boldsymbol{r}_0)=0$$

$$n_x(x-x_0)+n_y(y-y_0)+n_z(z-z_0)=0$$
$$ax+by+cz+d=0$$
$$(x, y, z, 1)\cdot(a, b, c, d)=0$$

Ha $\boldsymbol{n}$ egységvektor:

$\boldsymbol{n}\cdot(\boldsymbol{r}-\boldsymbol{r}_0)$ a síktól mért előjeles távolság!



Vektor/Pont/Sík/RGBA osztály

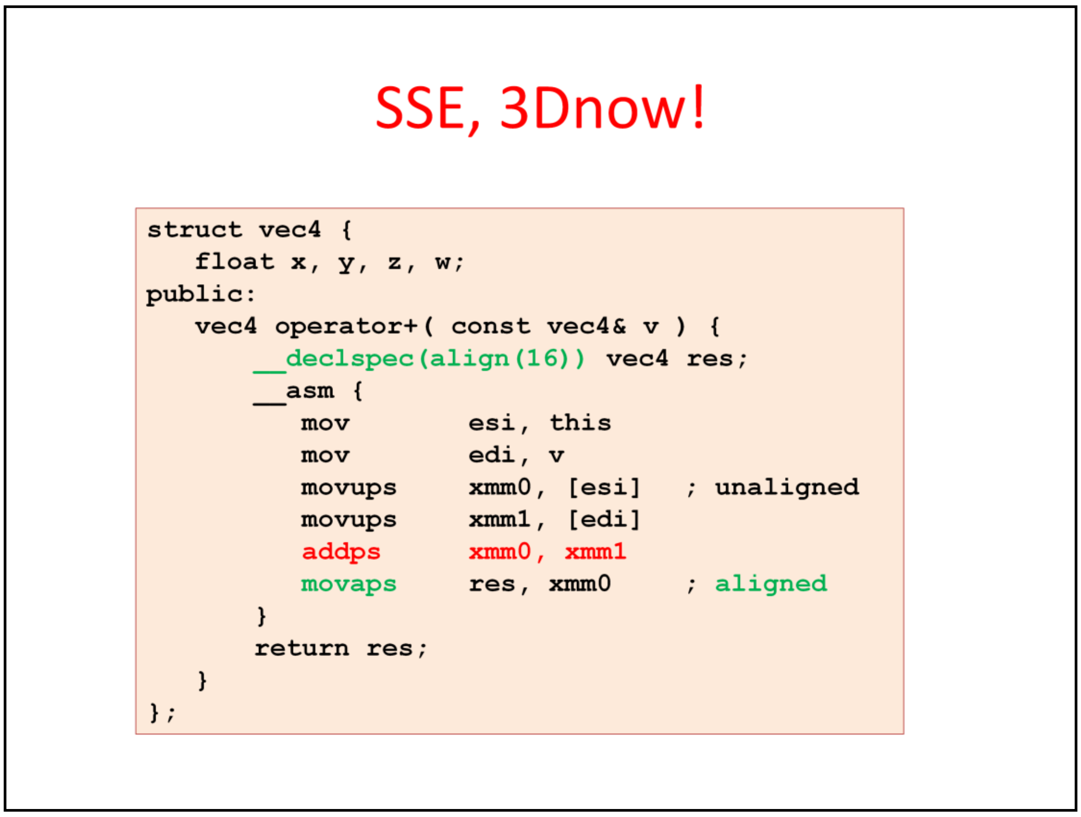
```
struct vec4 {
   float x, y, z, w;
   vec4(float x0, float y0, float z0, float w0) {
        x = x0; y = y0; z = z0;
        w = w0; // vektor:0, pont: 1, sík: d, RGBA: Opacitás
   }
   vec4 operator*(float a) { return vec4(x * a, y * a, z * a, w * a); }

   vec4 operator+(const vec4& v) {
        return vec4(x + v.x, y + v.y, z + v.z, w + v.w);
   }
   vec4 operator-(const vec4& v) {
        return vec4(x - v.x, y - v.y, z - v.z, w - v.w);
   }
   vec4 operator*(const vec4& v) {
        return vec4(x * v.x, y * v.y, z * v.z, w * v.w);
   }
};

float dot(const vec4& v1, const vec4& v2) {
   return (v1.x * v2.x + v1.y * v2.y + v1.z * v2.z + v1.w * v2.w);
}

float length(const vec4& v) { return sqrtf(dot(v, v)); }
```

Using vec4, i.e. four-element vectors, instead of vec3, we can represent not only points and vectors, but also planes. The fourth element should be set depending on the actual type.

Note that the test whether or not a point is on a plane is a dot product of the two 4-element vectors if the fourth component of a point is 1 and the four components of a plane is a, b, c, d, the plane parameters.

We can preserve the validity of vector operations if the fourth element of a vector is zero.

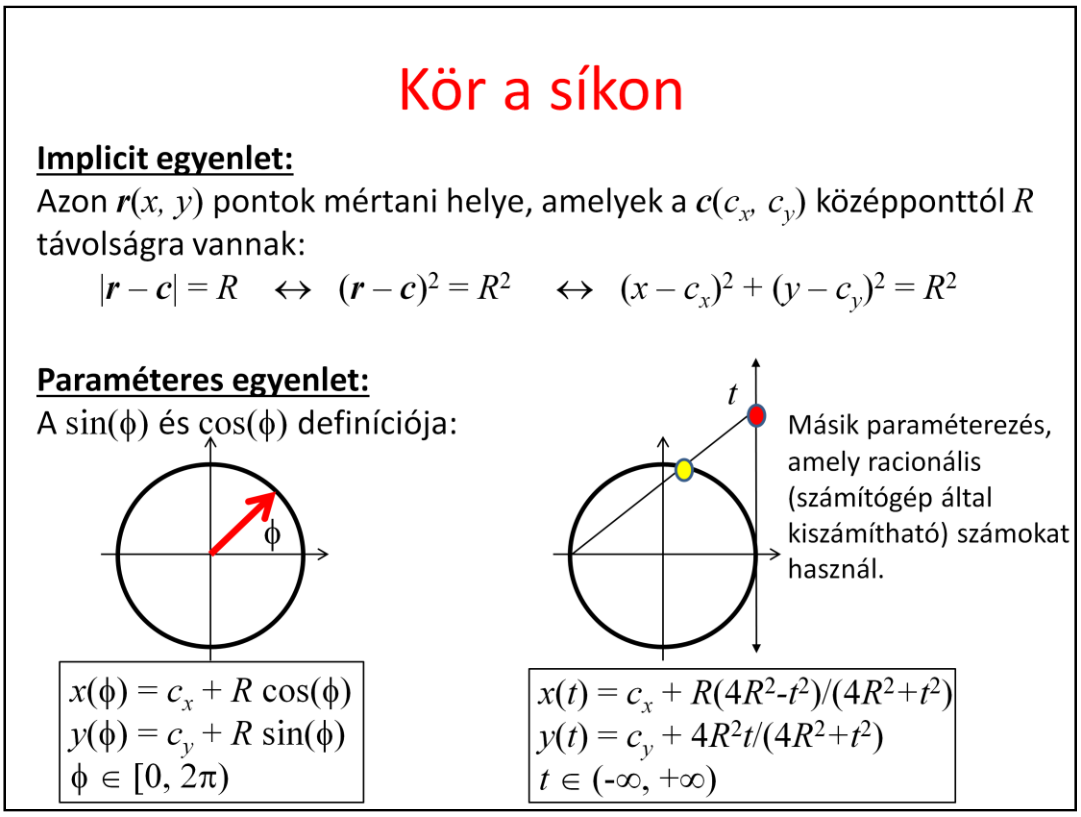
And now, the big news: those operations that are applicable for points remain valid even when the fourth element is 1.


SSE, 3Dnow!

```
struct vec4 {
   float x, y, z, w;
public:
   vec4 operator+( const vec4& v ) {
       __declspec(align(16)) vec4 res;
       __asm {
          mov        esi, this
          mov        edi, v
          movups     xmm0, [esi]   ; unaligned
          movups     xmm1, [edi]
          addps      xmm0, xmm1
          movaps     res, xmm0     ; aligned
       }
       return res;
   }
};
```

This concept is really important, so processor vendors developed special machine instructions for the efficient processing of four-element vectors. This is called **SSE = Streamed SIMD Extension**, where SIMD = Single Instruction Multiple Data.

Here you can see how two four float element vectors are added by a single machine instruction.



Kör a síkon

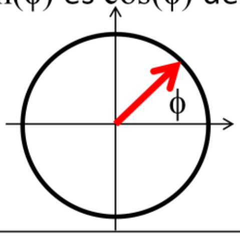
Implicit egyenlet:

Azon $\boldsymbol{r}(x, y)$ pontok mértani helye, amelyek a $\boldsymbol{c}(c_x, c_y)$ középponttól R távolságra vannak:

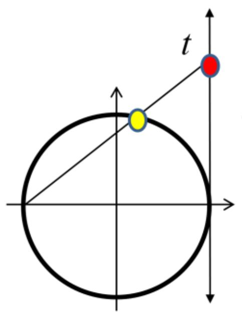
$$|\boldsymbol{r} - \boldsymbol{c}| = R \quad \leftrightarrow \quad (\boldsymbol{r} - \boldsymbol{c})^2 = R^2 \quad \leftrightarrow \quad (x - c_x)^2 + (y - c_y)^2 = R^2$$

Paraméteres egyenlet:

A sin(ϕ) és cos(ϕ) definíciója:



$$x(\phi) = c_x + R\cos(\phi)$$
$$y(\phi) = c_y + R\sin(\phi)$$
$$\phi \in [0, 2\pi)$$



Másik paraméterezés, amely racionális (számítógép által kiszámítható) számokat használ.

$$x(t) = c_x + R(4R^2 - t^2)/(4R^2 + t^2)$$
$$y(t) = c_y + 4R^2 t/(4R^2 + t^2)$$
$$t \in (-\infty, +\infty)$$

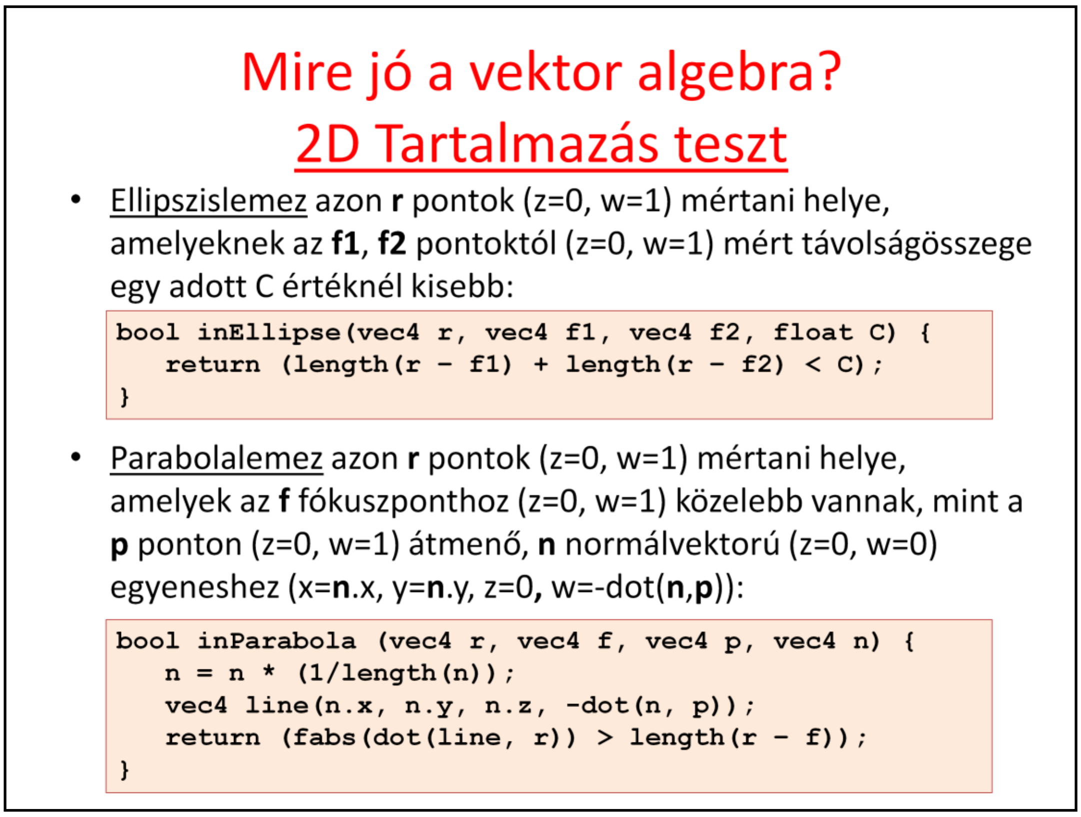
By definition, a **circle** is a set of points r of distance R (radius) from its center point c. Translating this geometric definition to the language of analytic geometry, we can establish the equation of the circle.

Distance of two points is the absolute value of the vector between them, which must be equal to R. Instead of the distance, we can work with the squared distance since both sides of this equation are positive, so taking the square does not modify the roots. The squared distance is the dot product of the difference vector with itself. Dot product can also be expressed with coordinates, so we can establish an implicit equation of the circle in Cartesian coordinates.

Circle has also a famous parametric equation, which is based on the definition of cos and sin: If we rotate a unit vector by ϕ around axis z, the x coordinate of the rotated vector is cos(ϕ) and the y coordinate is sin(ϕ).

Rotated vector of length R can be obtained by scaling by R. If the center is not in the origin but at point c, then we should translate the circle points by c.

As sine or cosine are usually irrational numbers, they cannot be precisely computed by a computer. So, sometimes another parametrization using only elementary operations is also useful (right hand side).



Mire jó a vektor algebra?
2D Tartalmazás teszt

- Ellipszislemez azon **r** pontok (z=0, w=1) mértani helye, amelyeknek az **f1**, **f2** pontoktól (z=0, w=1) mért távolságösszege egy adott C értéknél kisebb:

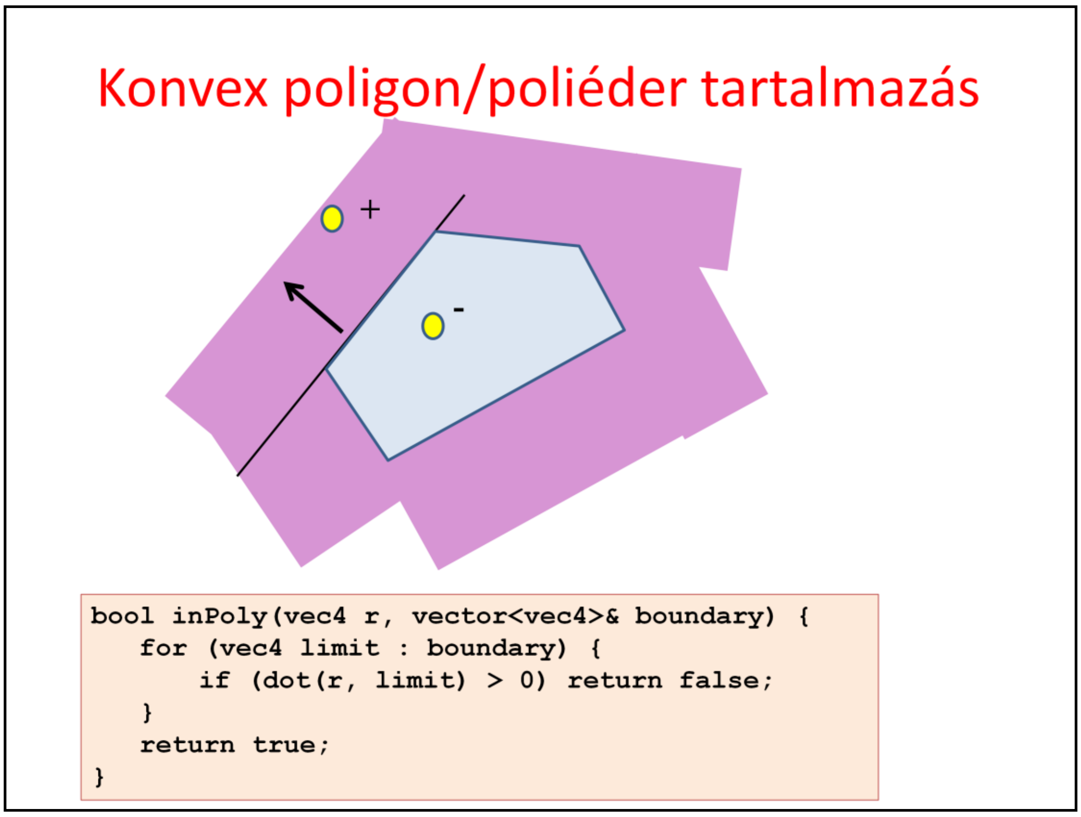
```
bool inEllipse(vec4 r, vec4 f1, vec4 f2, float C) {
   return (length(r – f1) + length(r – f2) < C);
}
```

- Parabolalemez azon **r** pontok (z=0, w=1) mértani helye, amelyek az **f** fókuszponthoz (z=0, w=1) közelebb vannak, mint a **p** ponton (z=0, w=1) átmenő, **n** normálvektorú (z=0, w=0) egyeneshez (x=**n**.x, y=**n**.y, z=0**,** w=-dot(**n**,**p**)):

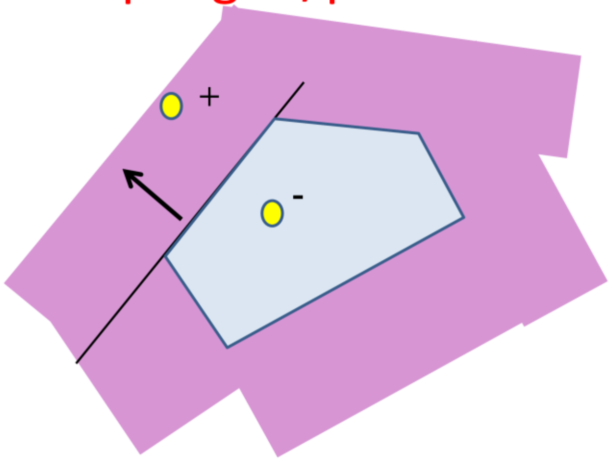
```
bool inParabola (vec4 r, vec4 f, vec4 p, vec4 n) {
   n = n * (1/length(n));
   vec4 line(n.x, n.y, n.z, -dot(n, p));
   return (fabs(dot(line, r)) > length(r – f));
}
```

We have concluded that a simple 2D rendering algorithm transforms pixel centers to world coordinates and checks which object includes the transformed point. It means that the essential operation of rendering is containment test.

Knowing vector algebra, containment test is easy to implement, we have to translate the definition of the object to vector algebra, and we are done.

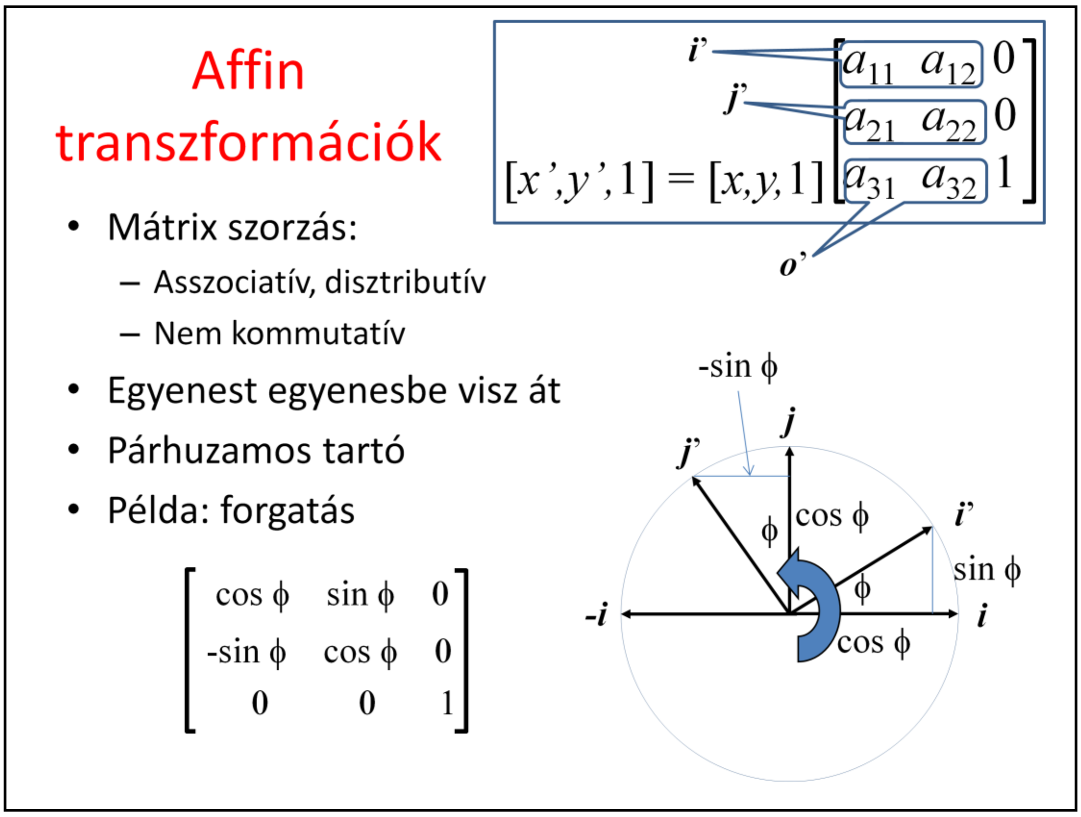


Konvex poligon/poliéder tartalmazás



```
bool inPoly(vec4 r, vector<vec4>& boundary) {
   for (vec4 limit : boundary) {
       if (dot(r, limit) > 0) return false;
   }
   return true;
}
```

A convex polyhedron is the intersection of half-spaces, thus a point is inside the convex polyhedron if it is inside all half-spaces.

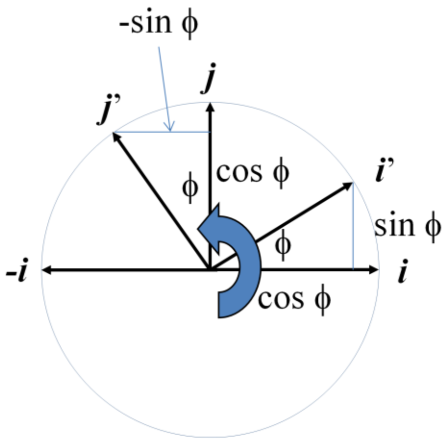


Affin transzformációk

$$\begin{array}{c} \\ i' \\ j' \\ o' \end{array} \quad [x', y', 1] = [x, y, 1] \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix}$$

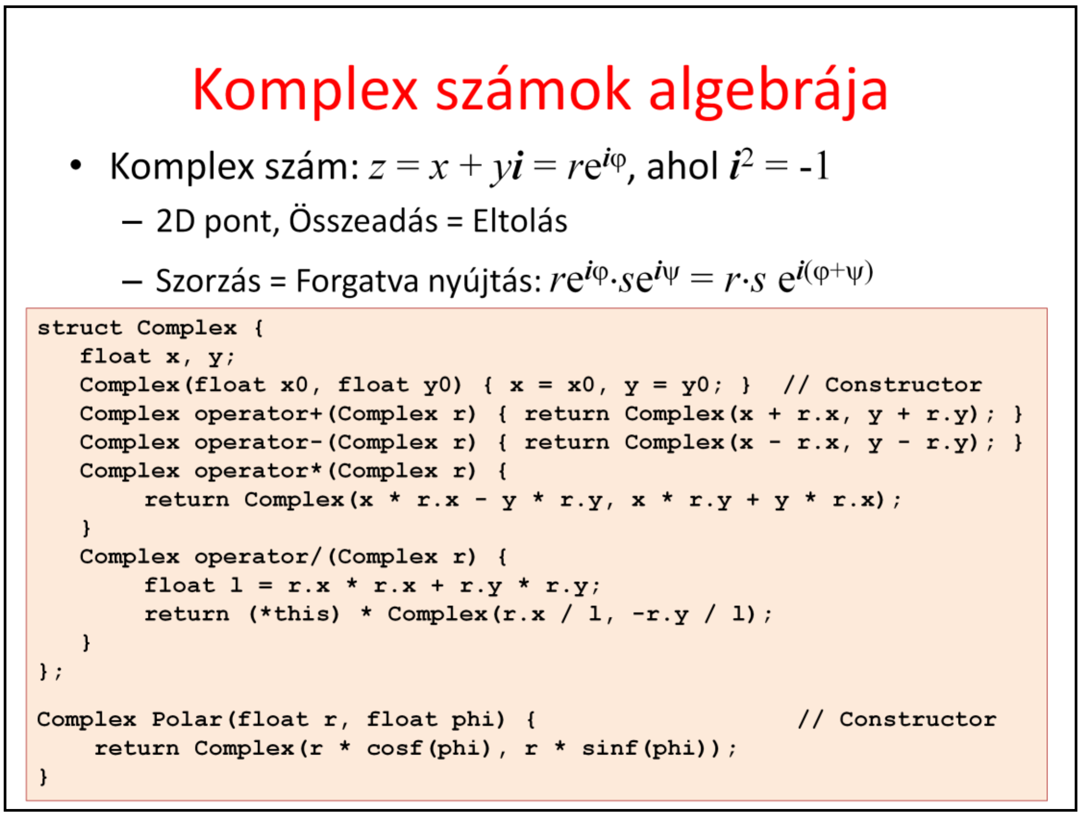
- Mátrix szorzás:
  - Asszociatív, disztributív
  - Nem kommutatív
- Egyenest egyenesbe visz át
- Párhuzamos tartó
- Példa: forgatás

$$\begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



In addition to vectors, we need other tools of algebra. Matrices are good for representing affine transformations. If matrices multiply vectors from the right, then the meaning of the rows of a matrix is the image of the basis vectors, i.e. points (1, 0) and (0,1), as well as the origin.

This concept is very important to us since we wish to find matrices executing certain transformations. To do this, we have to figure out how the basis vectors are modified by the transformation and the transformed vectors form the rows of the transformation matrix. As an example, we build a matrix that rotates around axis z by angle phi.



Komplex számok algebrája

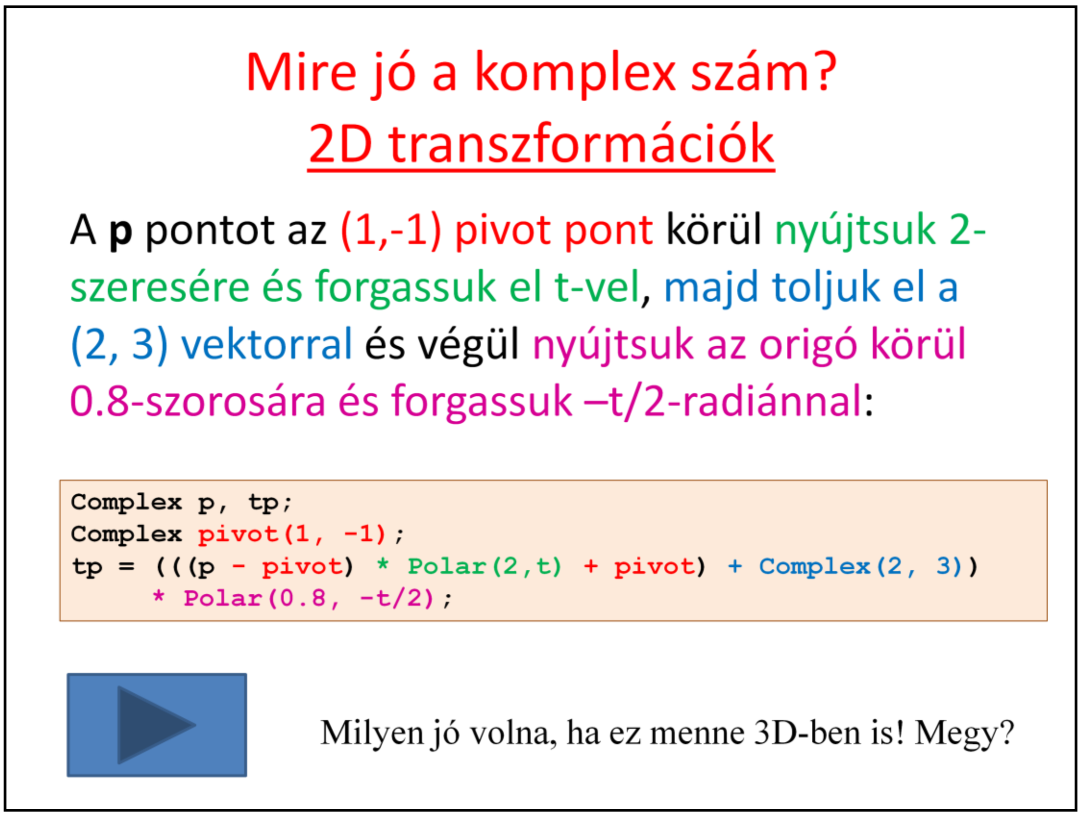
- Komplex szám: $z = x + y\boldsymbol{i} = re^{\boldsymbol{i}\varphi}$, ahol $\boldsymbol{i}^2 = -1$
  - 2D pont, Összeadás = Eltolás
  - Szorzás = Forgatva nyújtás: $re^{\boldsymbol{i}\varphi} \cdot se^{\boldsymbol{i}\psi} = r \cdot s\ e^{\boldsymbol{i}(\varphi+\psi)}$

```
struct Complex {
   float x, y;
   Complex(float x0, float y0) { x = x0, y = y0; }  // Constructor
   Complex operator+(Complex r) { return Complex(x + r.x, y + r.y); }
   Complex operator-(Complex r) { return Complex(x - r.x, y - r.y); }
   Complex operator*(Complex r) {
        return Complex(x * r.x - y * r.y, x * r.y + y * r.x);
   }
   Complex operator/(Complex r) {
        float l = r.x * r.x + r.y * r.y;
        return (*this) * Complex(r.x / l, -r.y / l);
   }
};

Complex Polar(float r, float phi) {                    // Constructor
    return Complex(r * cosf(phi), r * sinf(phi));
}
```

Vectors and matrices are not the only way to represent 2D points, translations and transformations. Complex numbers can do it all.

A complex number can represent a point. A complex number can also represent a translation vector, i.e. a translation if the vector's complex number is added to the point's complex number.

Finally, a complex number is also a rotation and scaling if multiplication is executed.



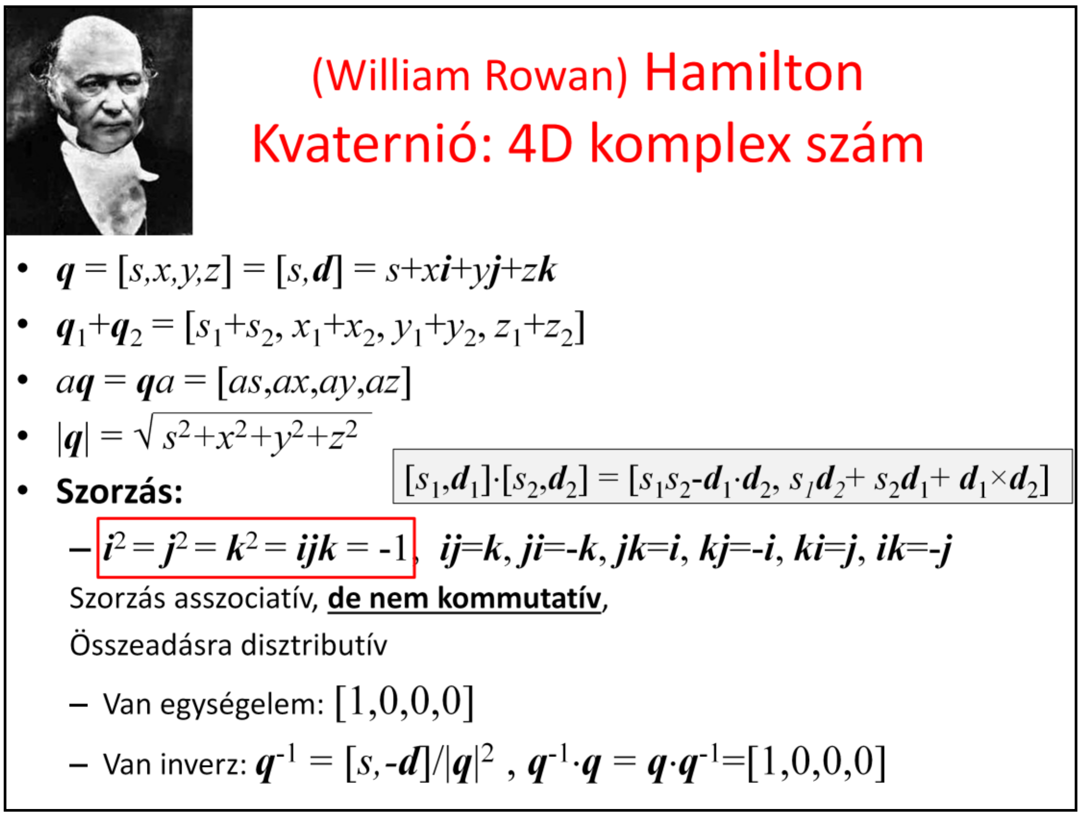
Mire jó a komplex szám?
2D transzformációk

A **p** pontot az (1,-1) pivot pont körül nyújtsuk 2-szeresére és forgassuk el t-vel, majd toljuk el a (2, 3) vektorral és végül nyújtsuk az origó körül 0.8-szorosára és forgassuk –t/2-radiánnal:

```
Complex p, tp;
Complex pivot(1, -1);
tp = (((p - pivot) * Polar(2,t) + pivot) + Complex(2, 3))
      * Polar(0.8, -t/2);
```

Milyen jó volna, ha ez menne 3D-ben is! Megy?

A C++ class implementing complex arithmetics can be used to implement a sequence of transformations.



(William Rowan) Hamilton Kvaternió: 4D komplex szám

- $\boldsymbol{q} = [s,x,y,z] = [s,\boldsymbol{d}] = s+x\boldsymbol{i}+y\boldsymbol{j}+z\boldsymbol{k}$
- $\boldsymbol{q}_1+\boldsymbol{q}_2 = [s_1+s_2, x_1+x_2, y_1+y_2, z_1+z_2]$
- $a\boldsymbol{q} = \boldsymbol{q}a = [as,ax,ay,az]$
- $|\boldsymbol{q}| = \sqrt{s^2+x^2+y^2+z^2}$
- **Szorzás:**
  - $\boxed{\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1}$, $\boldsymbol{ij}=\boldsymbol{k}$, $\boldsymbol{ji}=-\boldsymbol{k}$, $\boldsymbol{jk}=\boldsymbol{i}$, $\boldsymbol{kj}=-\boldsymbol{i}$, $\boldsymbol{ki}=\boldsymbol{j}$, $\boldsymbol{ik}=-\boldsymbol{j}$

    Szorzás asszociatív, **de nem kommutatív**,

    Összeadásra disztributív
  - Van egységelem: $[1,0,0,0]$
  - Van inverz: $\boldsymbol{q}^{-1} = [s,-\boldsymbol{d}]/|\boldsymbol{q}|^2$, $\boldsymbol{q}^{-1}\cdot\boldsymbol{q} = \boldsymbol{q}\cdot\boldsymbol{q}^{-1}=[1,0,0,0]$

$$\boxed{[s_1,\boldsymbol{d}_1]\cdot[s_2,\boldsymbol{d}_2] = [s_1s_2-\boldsymbol{d}_1\cdot\boldsymbol{d}_2,\ s_1\boldsymbol{d}_2+ s_2\boldsymbol{d}_1+ \boldsymbol{d}_1\times\boldsymbol{d}_2]}$$

Quaternions are 4D generalizations of complex number. Hamilton generalized them to 4D, because he realized that it is impossible to 3D without sacrificing the associative property of multiplication. Associativity is badly needed if we wish to represent rotations with quaternion multiplications since rotations, and generally

Mire jó a kvaternió: α szöggel az origón átmenő *d* irányú tengely körül

$\boldsymbol{q} = [\cos(\alpha/2), \boldsymbol{d} \sin(\alpha/2)], \quad |\boldsymbol{d}| = 1$

$\boldsymbol{q}\cdot[0,\boldsymbol{u}]\cdot\boldsymbol{q}^{-1} = [0,\boldsymbol{v}]$, $\boldsymbol{v}$ az $\boldsymbol{u}$ elforgatottja a $\boldsymbol{d}$ körül α -val

Példa: $\boldsymbol{u}=(1, 0, 0)$ z tengely körüli forgatása $\boldsymbol{d}=(0, 0, 1)$

$\boldsymbol{q}=[\cos(\alpha/2), 0, 0, \sin(\alpha/2)]\cdot[0,\boldsymbol{u}]\cdot[\cos(\alpha/2), 0, 0, -\sin(\alpha/2)]$

$= (\cos(\alpha/2)+\sin(\alpha/2)\boldsymbol{k}) \cdot \boldsymbol{i} \cdot (\cos(\alpha/2) - \sin(\alpha/2)\boldsymbol{k})$

$= (\cos(\alpha/2)\boldsymbol{i} +\sin(\alpha/2)\boldsymbol{j}) \cdot (\cos(\alpha/2) - \sin(\alpha/2)\boldsymbol{k})$

$= (\cos^2(\alpha/2)\boldsymbol{i} +\sin(\alpha/2)\cos(\alpha/2)\boldsymbol{j}+\cos(\alpha/2)\sin(\alpha/2)\boldsymbol{j}-\sin^2(\alpha/2)\boldsymbol{i}$

$= (\cos^2(\alpha/2)-\sin^2(\alpha/2))\boldsymbol{i} + 2\sin(\alpha/2)\cos(\alpha/2)\boldsymbol{j}$

$= \cos(\alpha)\boldsymbol{i} + \sin(\alpha)\boldsymbol{j}$

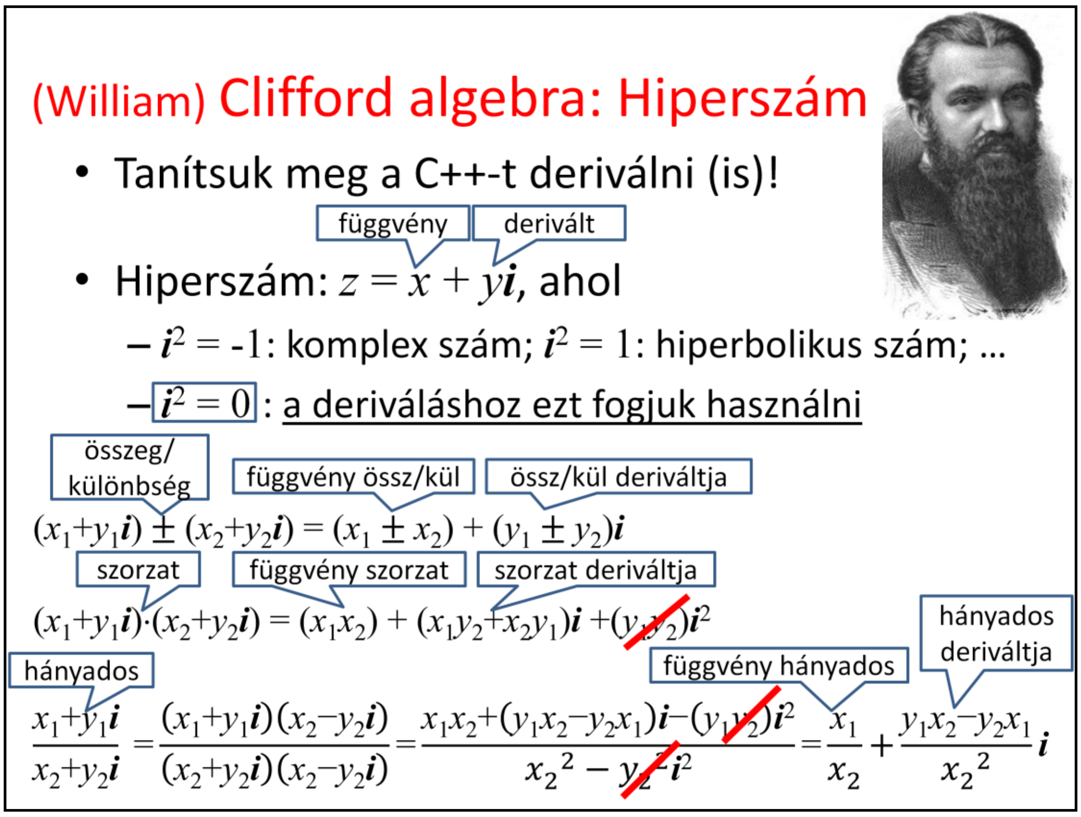
Implementáció: $\boldsymbol{q} = s+x\boldsymbol{i}+y\boldsymbol{j}+z\boldsymbol{k}$ = vec4

```
struct vec4 {
   float x, y, z, w;
   …
};
vec4 qmul(vec4 q1, vec4 q2) { // kvaternió szorzás
   vec3 d1(q1.x, q1.y, q1.z), d2(q2.x, q2.y, q2.z);
   return vec4(d2 * q1.w + d1 * q2.w + cross(d1, d2),
               q1.w * q2.w - dot(d1, d2));
}
vec4 quaternion(float ang, vec3 axis) { // konstruálás
   vec3 d = normalize(axis) * sinf(ang / 2);
   return vec4(d.x, d.y, d.z, cosf(ang / 2));
}
vec3 Rotate(vec3 u, vec4 q) {
   vec4 qinv(-q.x, -q.y, -q.z, q.w);
   vec4 qr = qmul(qmul(q, vec4(u, 0)), qinv);
   return vec3(qr.x, qr.y, qr.z);
}
```

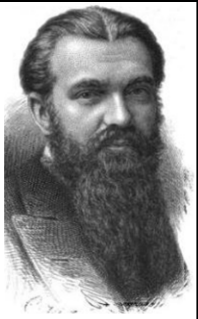
Vigyázat:
GPU shader programozás GLSL nyelven!

```
uniform vec4 q;    // quaternion as uniform variable
in vec3 u;         // Varying input: vertex

vec4 qmul(vec4 q1, vec4 q2) {
  return vec4(
    q1.w * q2.xyz + q2.w * q1.xyz + cross(q1.xyz, q2.xyz),
    q1.w * q2.w - dot(q1.xyz, q2.xyz));
}

void main() { // vertex shader program
  vec4 qinv = vec4(-q.xyz,q.w);
  vec3 v = qmul(qmul(q, vec4(u, 0)), qinv).xyz;
  gl_Position = vec4(v, 1);
}
```

Deriválni tudni necesse est,
vivere non est necesse.



(William) Clifford algebra: Hiperszám



- Tanítsuk meg a C++-t deriválni (is)!
- Hiperszám: $z = \underbrace{x}_{\text{függvény}} + \underbrace{y}_{\text{derivált}}\boldsymbol{i}$, ahol
  - $\boldsymbol{i}^2 = -1$: komplex szám; $\boldsymbol{i}^2 = 1$: hiperbolikus szám; …
  - $\boxed{\boldsymbol{i}^2 = 0}$: a deriváláshoz ezt fogjuk használni

összeg/ különbség — függvény össz/kül — össz/kül deriváltja

$$(x_1+y_1\boldsymbol{i}) \pm (x_2+y_2\boldsymbol{i}) = (x_1 \pm x_2) + (y_1 \pm y_2)\boldsymbol{i}$$

szorzat — függvény szorzat — szorzat deriváltja

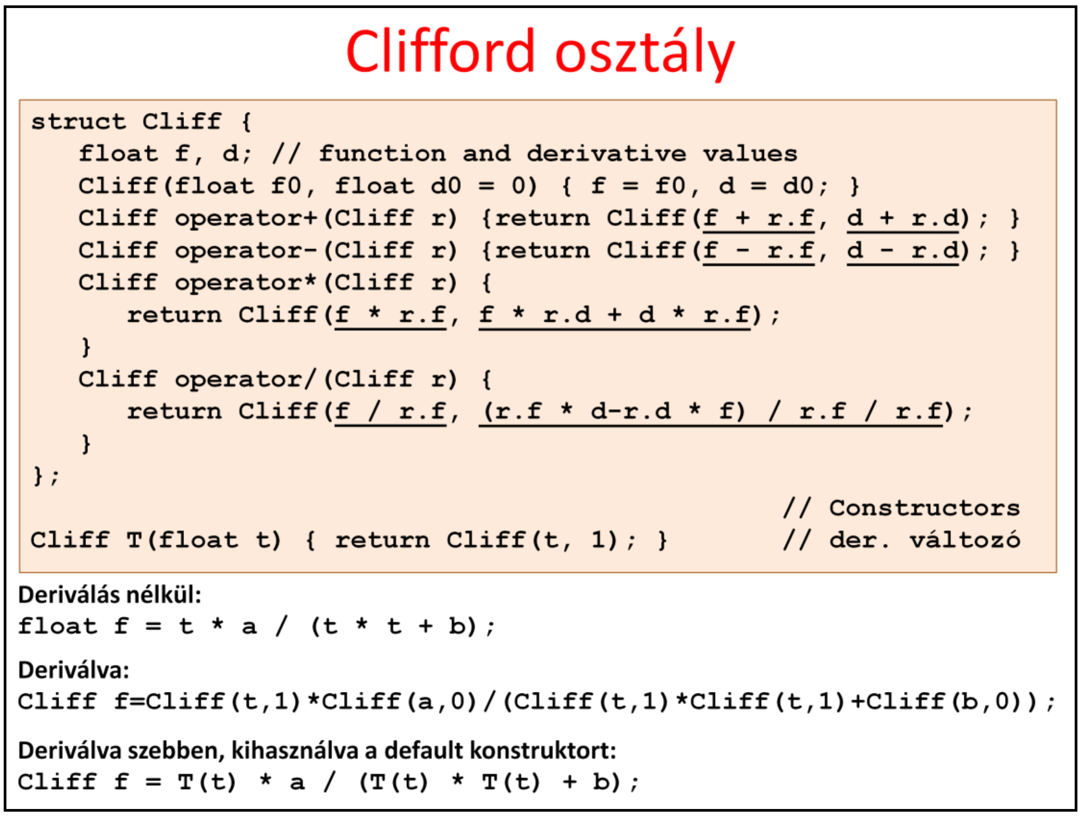
$$(x_1+y_1\boldsymbol{i})\cdot(x_2+y_2\boldsymbol{i}) = (x_1x_2) + (x_1y_2+x_2y_1)\boldsymbol{i} + \cancel{(y_1y_2)}\boldsymbol{i}^2$$

hányados — függvény hányados — hányados deriváltja

$$\frac{x_1+y_1\boldsymbol{i}}{x_2+y_2\boldsymbol{i}} = \frac{(x_1+y_1\boldsymbol{i})(x_2-y_2\boldsymbol{i})}{(x_2+y_2\boldsymbol{i})(x_2-y_2\boldsymbol{i})} = \frac{x_1x_2+(y_1x_2-y_2x_1)\boldsymbol{i}-\cancel{(y_1y_2)}\boldsymbol{i}^2}{x_2^2-\cancel{y_2^2}\boldsymbol{i}^2} = \frac{x_1}{x_2} + \frac{y_1x_2-y_2x_1}{x_2^2}\boldsymbol{i}$$

The set of complex numbers is a special case of hyper numbers of Clifford algebras that deal with numbers of form x+yi where x and y are real numbers and i is a symbol, and the same arithmetic rules apply for such numbers as for real ones. To use the arithmetic rules, we should decide what is the value of i^2. Complex numbers define it as -1, in Clifford algebra, this is up to us.

If we take the option of i^2=0, then the rules will be equivalent to the rules of derivatives assuming x the function value and y the derivative. Thus, we can derivate without approximations specifying only the function, the derivative is obtained automatically.



Clifford osztály

```
struct Cliff {
   float f, d; // function and derivative values
   Cliff(float f0, float d0 = 0) { f = f0, d = d0; }
   Cliff operator+(Cliff r) {return Cliff(f + r.f, d + r.d); }
   Cliff operator-(Cliff r) {return Cliff(f - r.f, d - r.d); }
   Cliff operator*(Cliff r) {
      return Cliff(f * r.f, f * r.d + d * r.f);
   }
   Cliff operator/(Cliff r) {
      return Cliff(f / r.f, (r.f * d-r.d * f) / r.f / r.f);
   }
};
                                                 // Constructors
Cliff T(float t) { return Cliff(t, 1); }        // der. változó
```

Deriválás nélkül:

```
float f = t * a / (t * t + b);
```

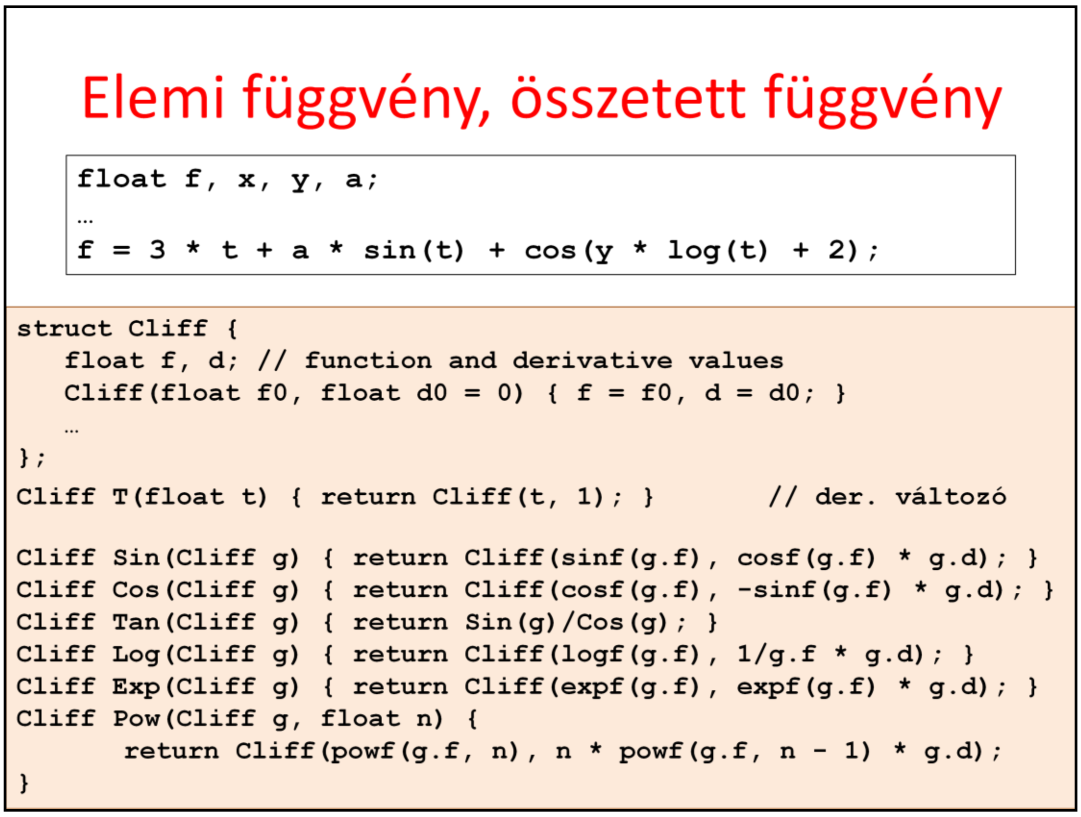
Deriválva:

```
Cliff f=Cliff(t,1)*Cliff(a,0)/(Cliff(t,1)*Cliff(t,1)+Cliff(b,0));
```

Deriválva szebben, kihasználva a default konstruktort:

```
Cliff f = T(t) * a / (T(t) * T(t) + b);
```

Cliff implements the arithmetic rules of derivation.

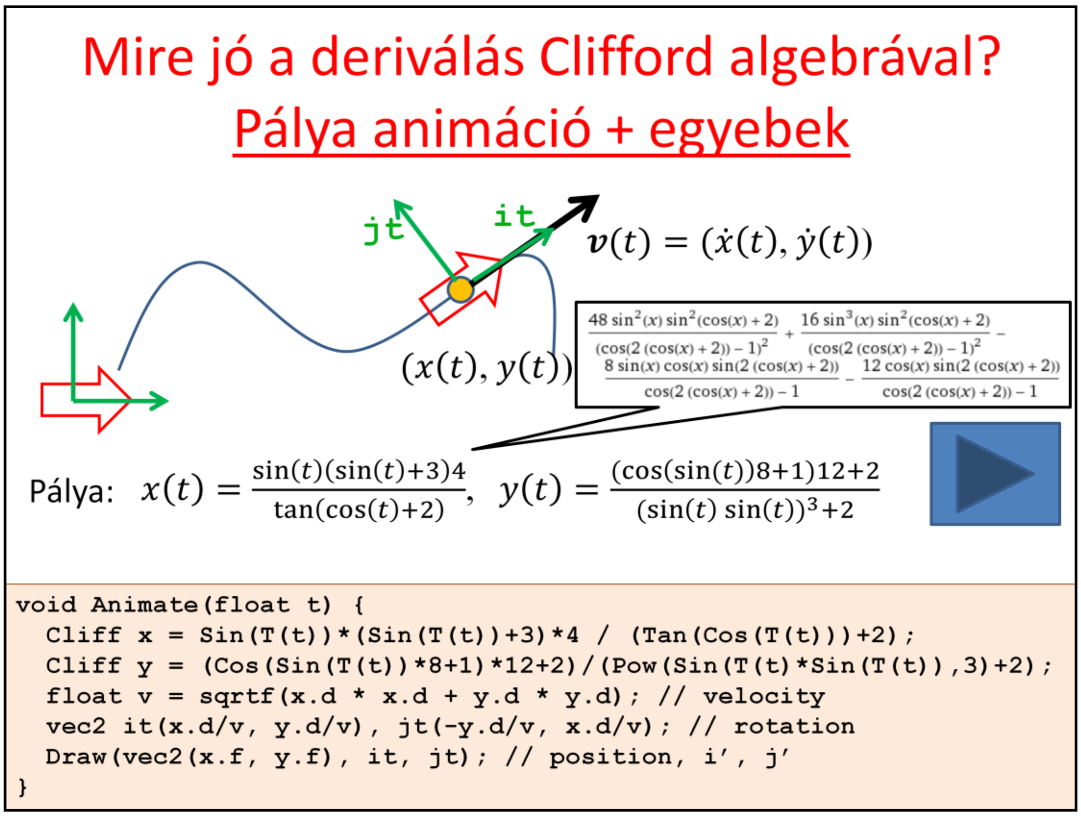


Elemi függvény, összetett függvény

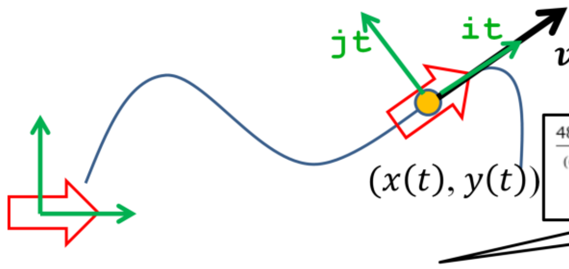
```
float f, x, y, a;
…
f = 3 * t + a * sin(t) + cos(y * log(t) + 2);
```

```
struct Cliff {
   float f, d; // function and derivative values
   Cliff(float f0, float d0 = 0) { f = f0, d = d0; }
   …
};
Cliff T(float t) { return Cliff(t, 1); }        // der. változó

Cliff Sin(Cliff g) { return Cliff(sinf(g.f), cosf(g.f) * g.d); }
Cliff Cos(Cliff g) { return Cliff(cosf(g.f), -sinf(g.f) * g.d); }
Cliff Tan(Cliff g) { return Sin(g)/Cos(g); }
Cliff Log(Cliff g) { return Cliff(logf(g.f), 1/g.f * g.d); }
Cliff Exp(Cliff g) { return Cliff(expf(g.f), expf(g.f) * g.d); }
Cliff Pow(Cliff g, float n) {
      return Cliff(powf(g.f, n), n * powf(g.f, n - 1) * g.d);
}
```

We also need global functions to specify the derivatives of elementary functions.



Mire jó a deriválás Clifford algebrával? Pálya animáció + egyebek



$$v(t) = (\dot{x}(t), \dot{y}(t))$$

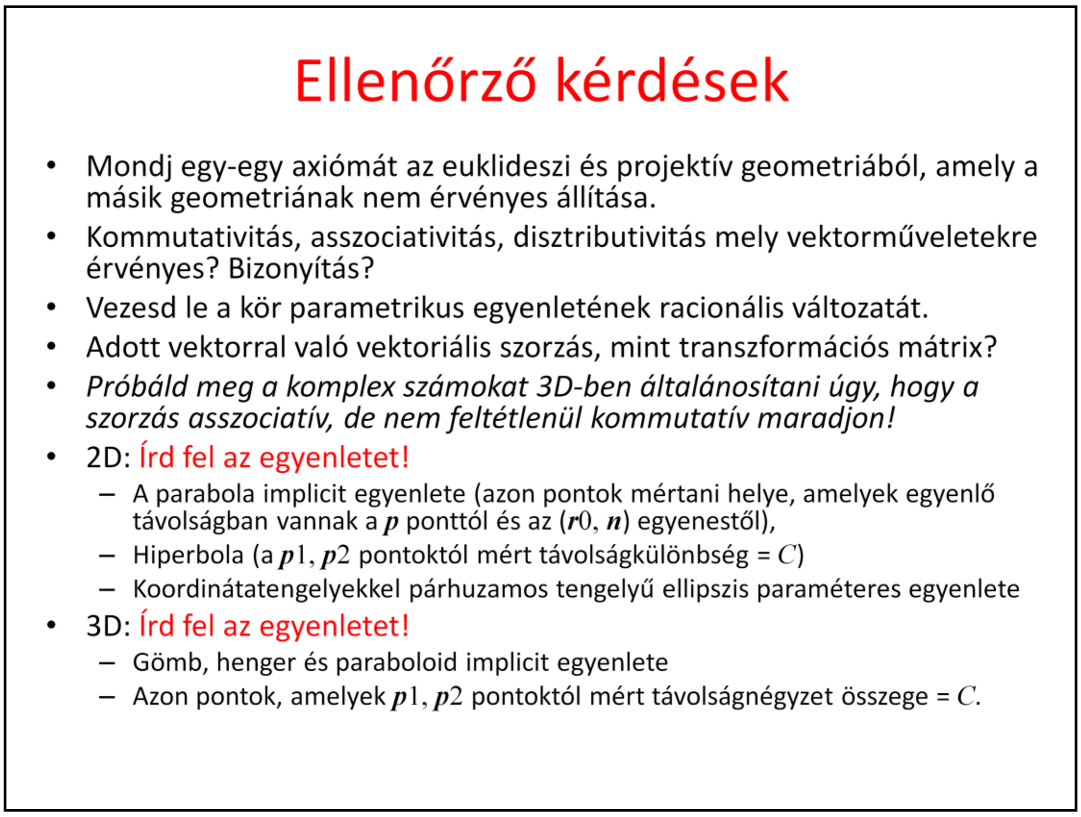
$$\frac{48\sin^2(x)\sin^2(\cos(x)+2)}{(\cos(2(\cos(x)+2))-1)^2} + \frac{16\sin^3(x)\sin^2(\cos(x)+2)}{(\cos(2(\cos(x)+2))-1)^2} - \frac{8\sin(x)\cos(x)\sin(2(\cos(x)+2))}{\cos(2(\cos(x)+2))-1} - \frac{12\cos(x)\sin(2(\cos(x)+2))}{\cos(2(\cos(x)+2))-1}$$

Pálya: $x(t) = \dfrac{\sin(t)(\sin(t)+3)4}{\tan(\cos(t)+2)}$, $y(t) = \dfrac{(\cos(\sin(t))8+1)12+2}{(\sin(t)\sin(t))^3+2}$

```
void Animate(float t) {
  Cliff x = Sin(T(t))*(Sin(T(t))+3)*4 / (Tan(Cos(T(t)))+2);
  Cliff y = (Cos(Sin(T(t))*8+1)*12+2)/(Pow(Sin(T(t)*Sin(T(t)),3)+2);
  float v = sqrtf(x.d * x.d + y.d * y.d); // velocity
  vec2 it(x.d/v, y.d/v), jt(-y.d/v, x.d/v); // rotation
  Draw(vec2(x.f, y.f), it, jt); // position, i', j'
}
```

Suppose we want to animate an object along a path defined by (x(t), y(t)). The position of the object is given by inserting time t into these two functions.

Objects like trains, cars, birds, etc. follow their front (head, beak) during animation, so we should often rotate the object to transform its heading direction into the current direction of movement, which is the current velocity, i.e. the derivative of the path.



Ellenőrző kérdések

- Mondj egy-egy axiómát az euklideszi és projektív geometriából, amely a másik geometriának nem érvényes állítása.
- Kommutativitás, asszociativitás, disztributivitás mely vektorműveletekre érvényes? Bizonyítás?
- Vezesd le a kör parametrikus egyenletének racionális változatát.
- Adott vektorral való vektoriális szorzás, mint transzformációs mátrix?
- *Próbáld meg a komplex számokat 3D-ben általánosítani úgy, hogy a szorzás asszociatív, de nem feltétlenül kommutatív maradjon!*
- 2D: Írd fel az egyenletet!
  - A parabola implicit egyenlete (azon pontok mértani helye, amelyek egyenlő távolságban vannak a $\boldsymbol{p}$ ponttól és az ($\boldsymbol{r}0$, $\boldsymbol{n}$) egyenestől),
  - Hiperbola (a $\boldsymbol{p}1$, $\boldsymbol{p}2$ pontoktól mért távolságkülönbség = C)
  - Koordinátatengelyekkel párhuzamos tengelyű ellipszis paraméteres egyenlete
- 3D: Írd fel az egyenletet!
  - Gömb, henger és paraboloid implicit egyenlete
  - Azon pontok, amelyek $\boldsymbol{p}1$, $\boldsymbol{p}2$ pontoktól mért távolságnégyzet összege = C.

This problems should be solved by you and by you alone to check whether or not the topic is understood. Some of them are trivial, others need effort. Do not get angry if you cannot solve all of them, the important thing is to try.

As reading the solutions is completely pointless, they are not here.