

“The only legitimate use of a computer is to play games.”

Eugene Jarvis

Játékfejlesztés

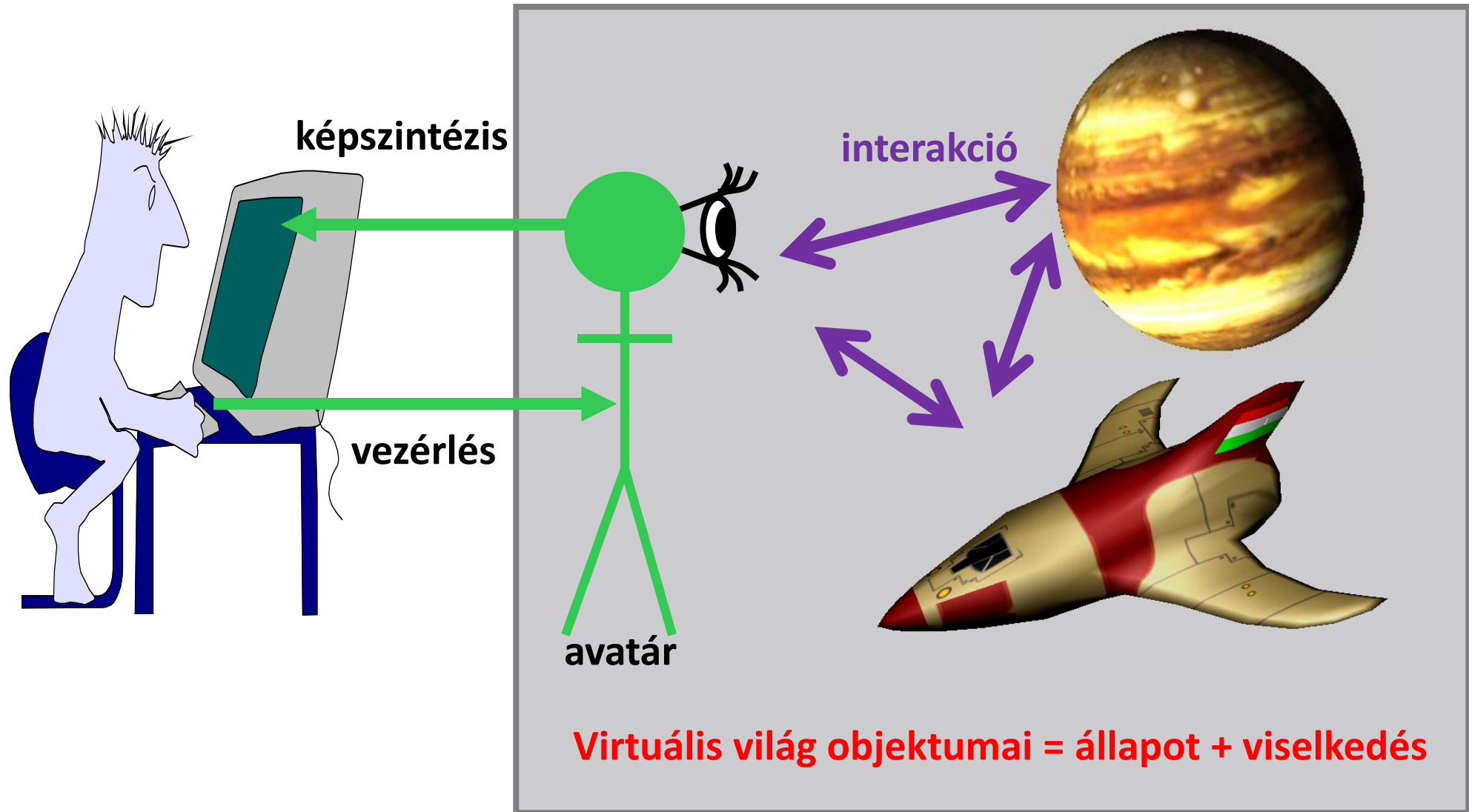
Szirmay-Kalos

László





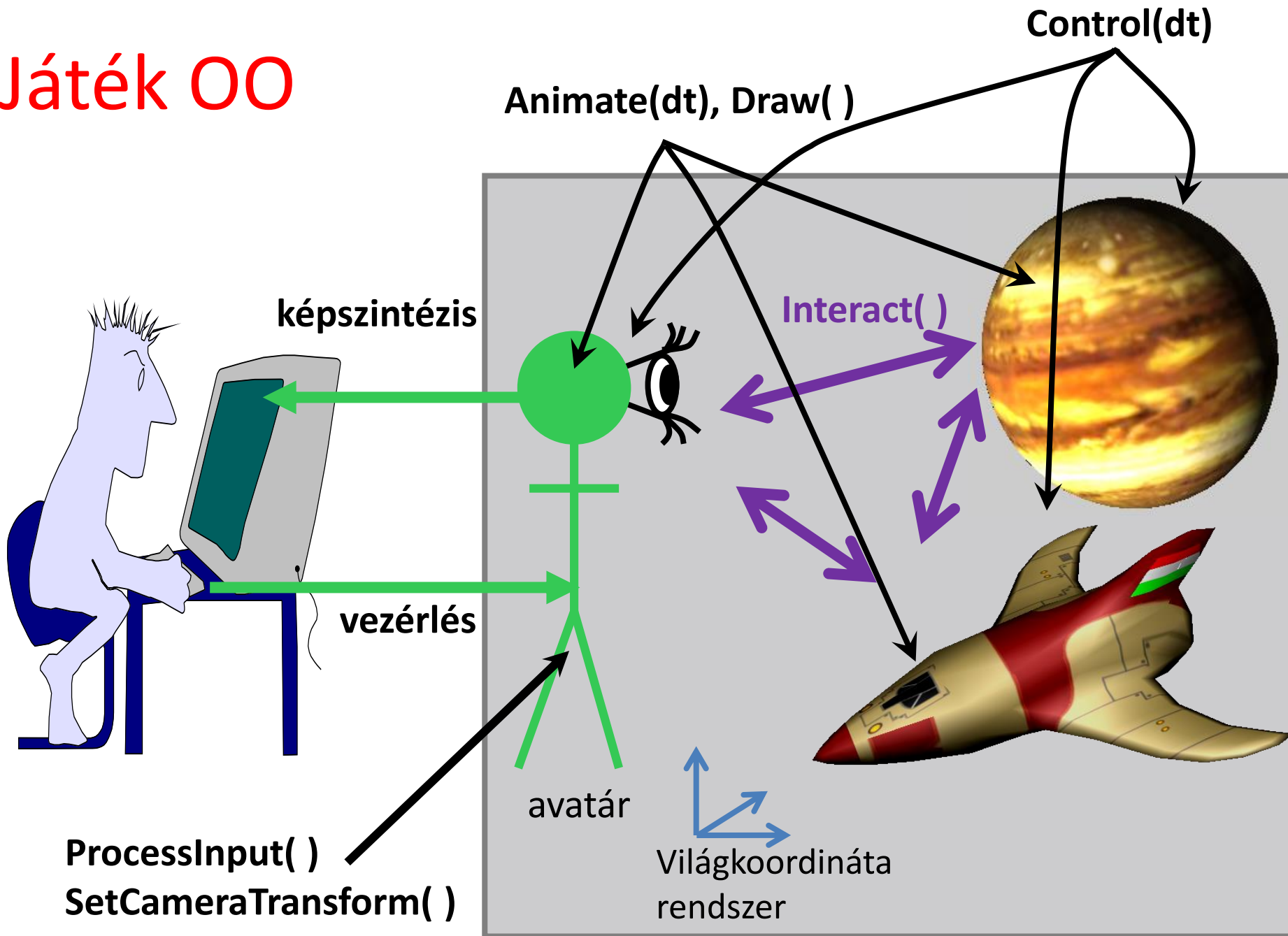
Virtuális valóság



Játékok feladatai

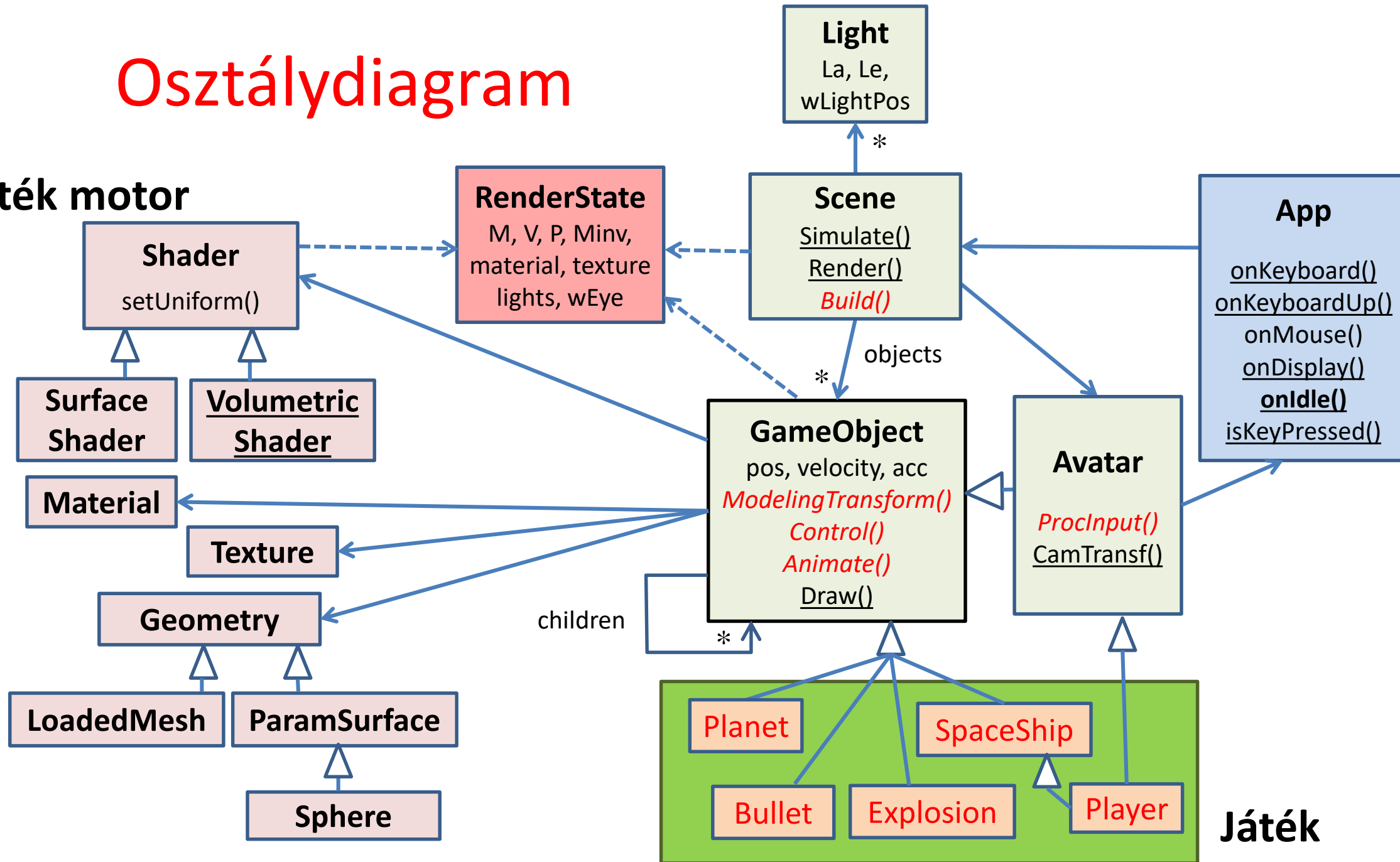
- Képszintézis az avatar nézőpontjából:
 - 1 szem, sztereo, autosztereo
- Az avatar vezérlése a beviteli eszközökkel:
 - keyboard, mouse, Wii, gépi látás, Kinect, VR sisak, stb.
- Az „intelligens” objektumok vezérlése (AI)
 - állapotgép
- A fizikai világ szimulációja
 - Newtoni fizika

Játék 00



Osztálydiagram

Játék motor

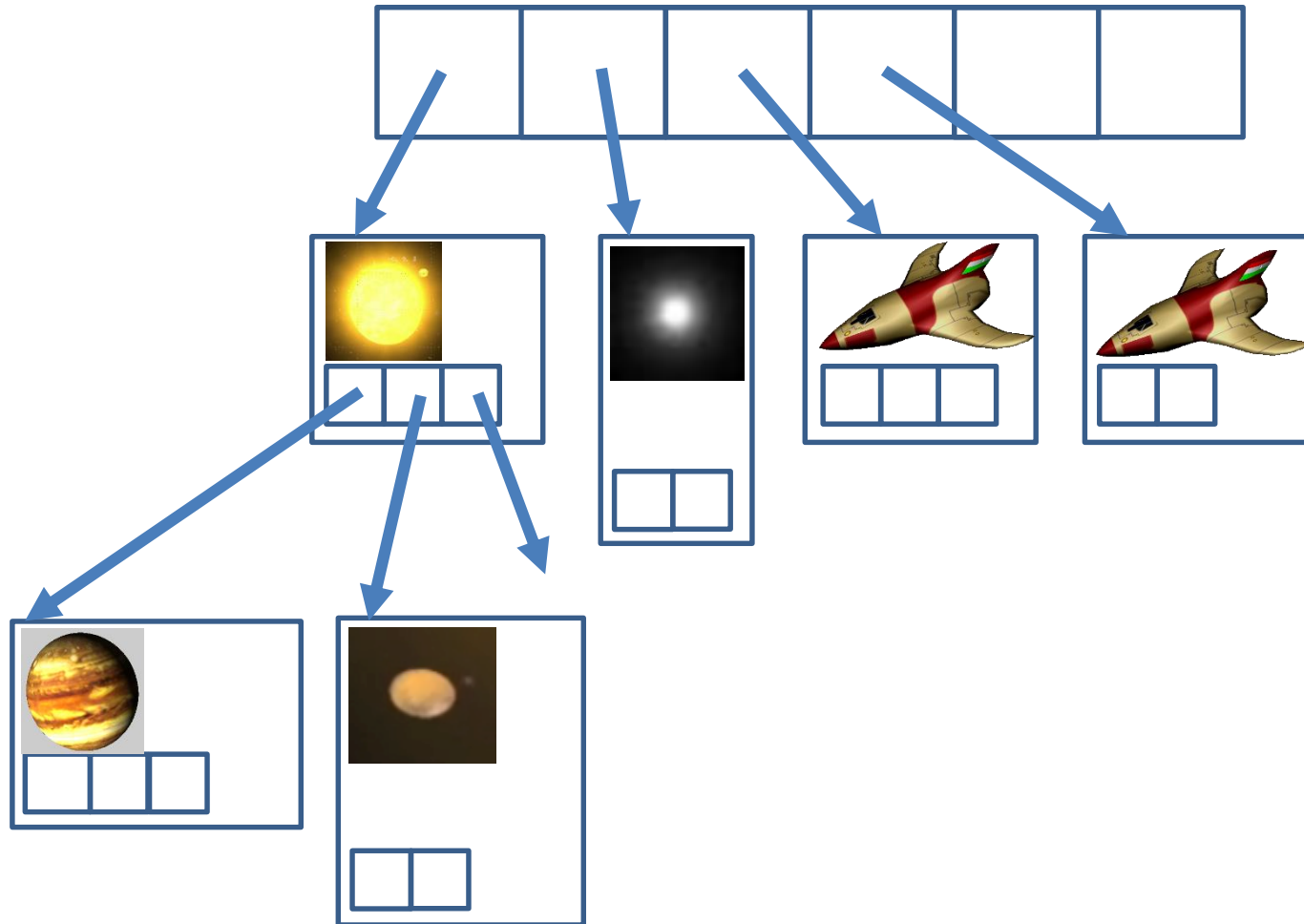


Játékobjektum (GameObject)

```
class GameObject {  
protected:  
    Shader *    shader;  
    Material * material;  
    Texture *   texture;  
    Geometry * geometry;  
    vec3 pos, velocity, acceleration;  
    vector<GameObject *> children;  
    virtual void ModelingTransform(RenderState& state) { }  
public:  
    GameObject(Shader* s, Material* m, Texture* t, Geometry* g) { ... }  
    virtual void Control(float dt) { }  
    virtual void Animate(float dt) { }  
    virtual void Draw(RenderState state) { // parameter by value to separate objects  
        ModelingTransform(state);  
        state.material = material; state.texture = texture;  
        shader->setUniform(state); // uniform variable setting  
        geometry->Draw(); // triangles go down the pipeline  
        for (Object * child : children) child->Draw(state);  
    }  
};
```

Virtuális világ (Scene)

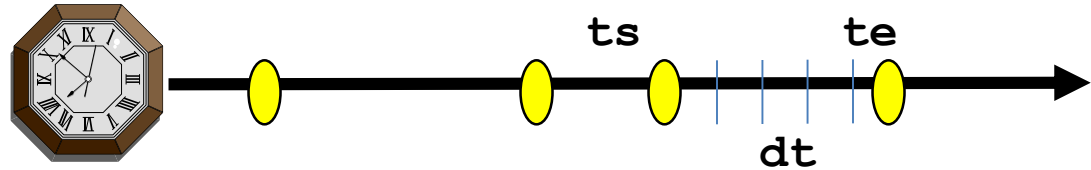
```
vector<GameObject *> objects;
```



Szimulációs hurok (Game loop)

```
void onIdle ( ) { // idle call back  
    static float tend = 0;  
    float tstart = tend;  
    tend = glutGet(GLUT_ELAPSED_TIME);  
    scene.Simulate(tstart, tend, keys);  
    glutPostRedisplay();  
}
```

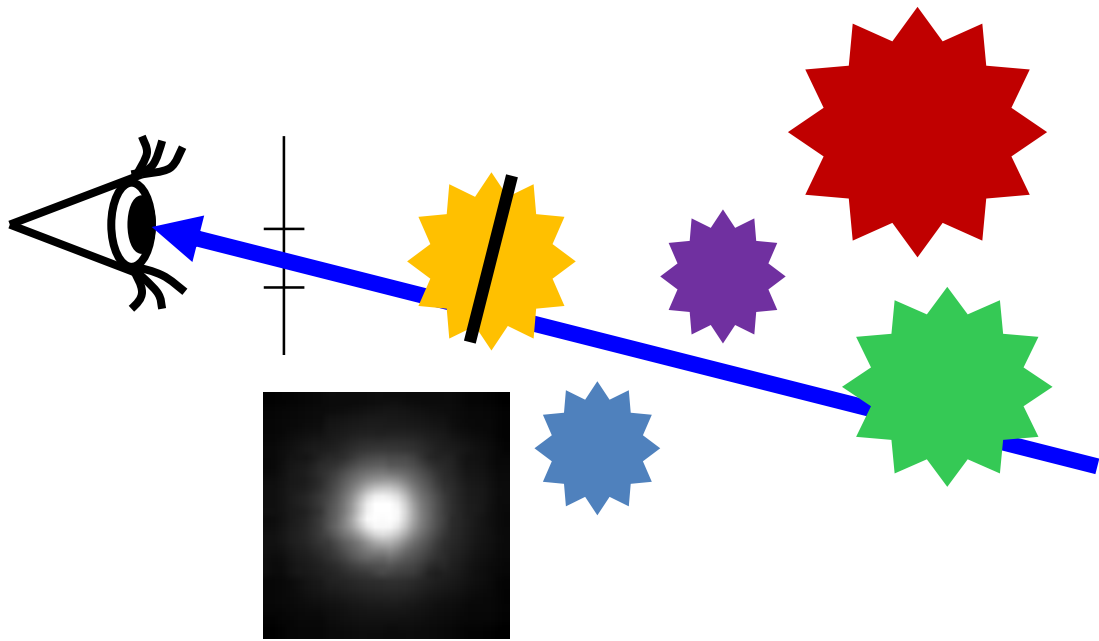
```
void onDisplay() {  
    glClear(GL_COLOR_BUFFER_BIT |  
           GL_DEPTH_BUFFER_BIT);  
    scene.Render();  
    glutSwapBuffers( );  
}
```



```
void Scene::Simulate(float ts, float te) {  
    avatar->ProcInput();  
    for(float t = ts; t < te; t += dt) {  
        float Dt = fmin(dt, te - t);  
        for (GameObject * o : objects) o->Control(Dt);  
        for (GameObject * o : objects) o->Animate(Dt);  
    }  
}
```

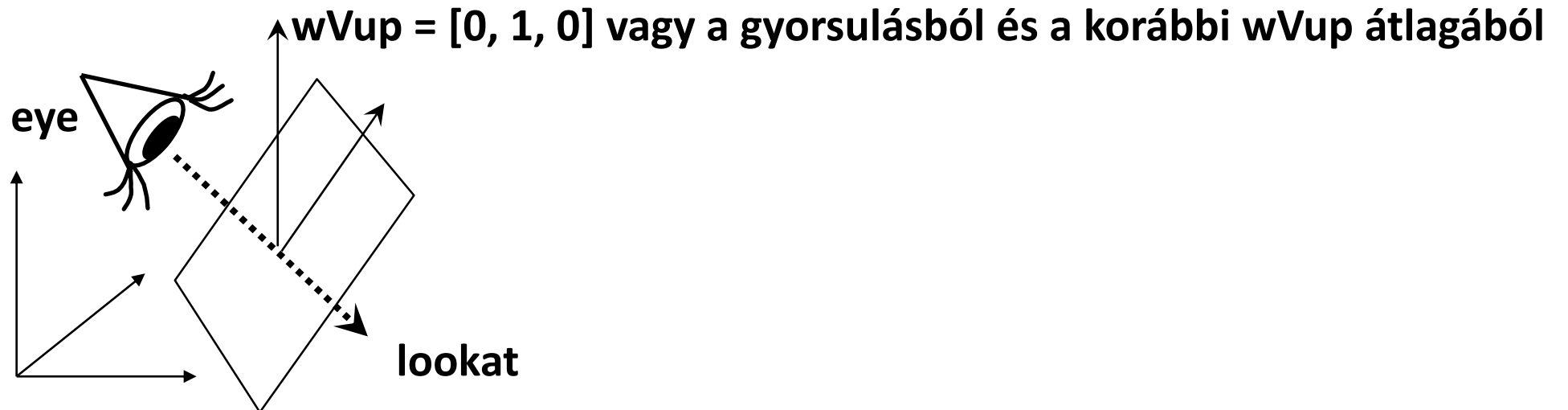
```
void Scene::Render( )  
{  
    RenderState state; // M=Minv=UnitMatrix();  
    avatar->SetCameraTransform(state);  
    for (GameObject * o : objects) o->Draw(state);  
}
```


Volumetric shader

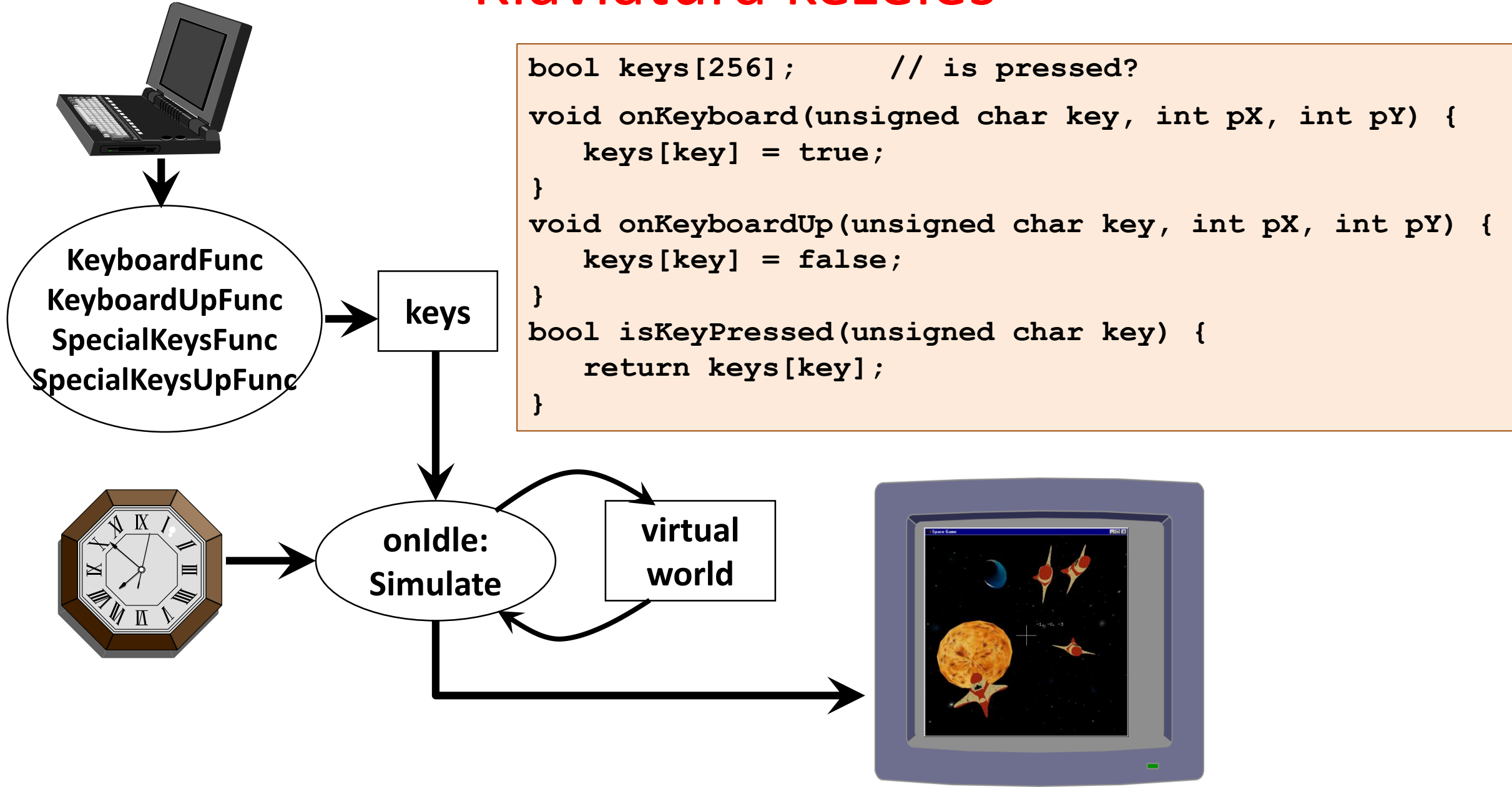


Avatar

```
struct Avatar : public GameObject {  
    virtual void ProcessInput() { }  
    virtual vec3 wVup() { return vec3(0, 1, 0); }  
    void SetCameraTransform(RenderState& state) {  
        Camera camera(pos, pos + velocity, wVup());  
        state.V() = camera.V();  
        state.P() = camera.P();  
    }  
};
```



Klaviatúra kezelés



```
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;  
}  
bool isKeyPressed(unsigned char key) {  
    return keys[key];  
}
```

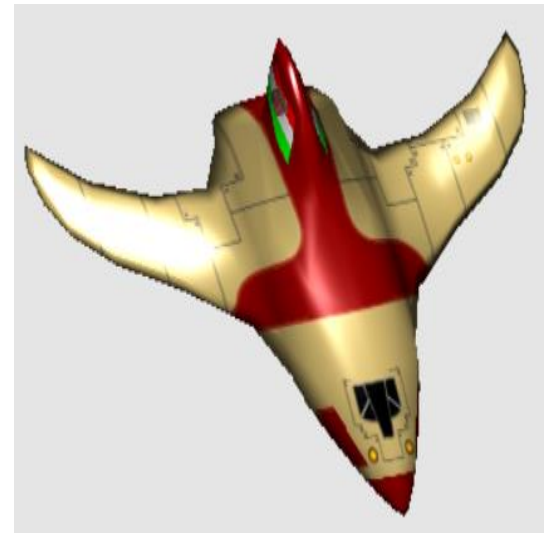
*“Be van fejezve a nagy mű, igen.
A gép forog, az alkotó pihen.
Évmilliókig eljár tengelyén,
Mig egy kerékfogát ujítani kell.”*

Madách Imre

Játékfejlesztés

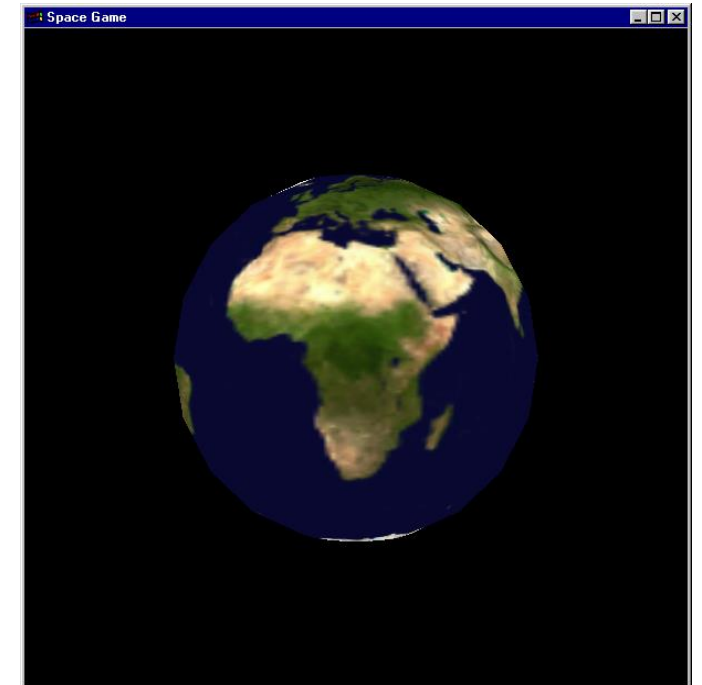
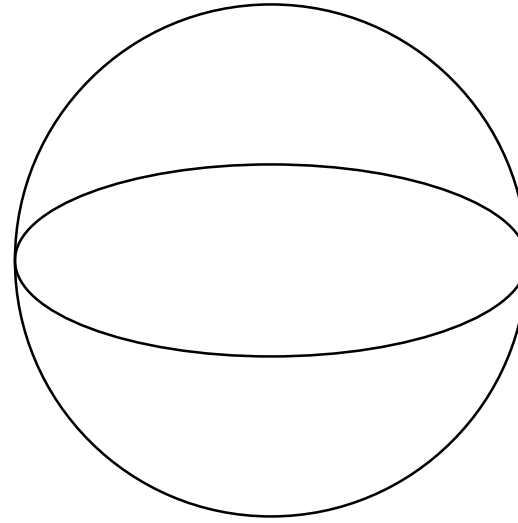
2. Játékobjektumok

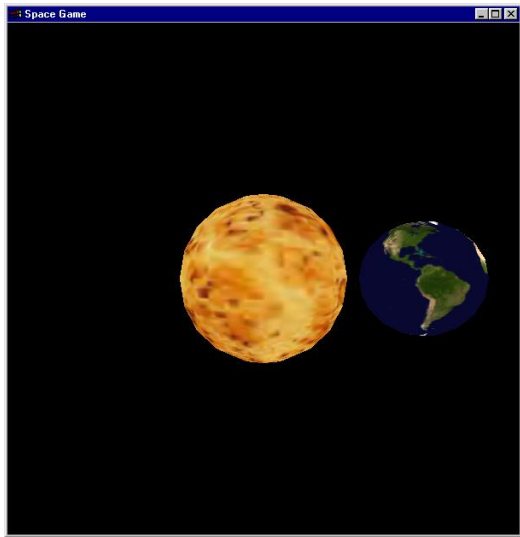
Szirmay-Kalos László



Bolygó: Planet

- Geometria: gömb
- Textúra
- Animáció:
 - Fizikai:
 - Tájékozódik majd követi a gravitációs törvényt
 - Képletanimáció:
 - „beégetett pálya”
 - Többiek érdektelenek
 - Nincs respektált törvény



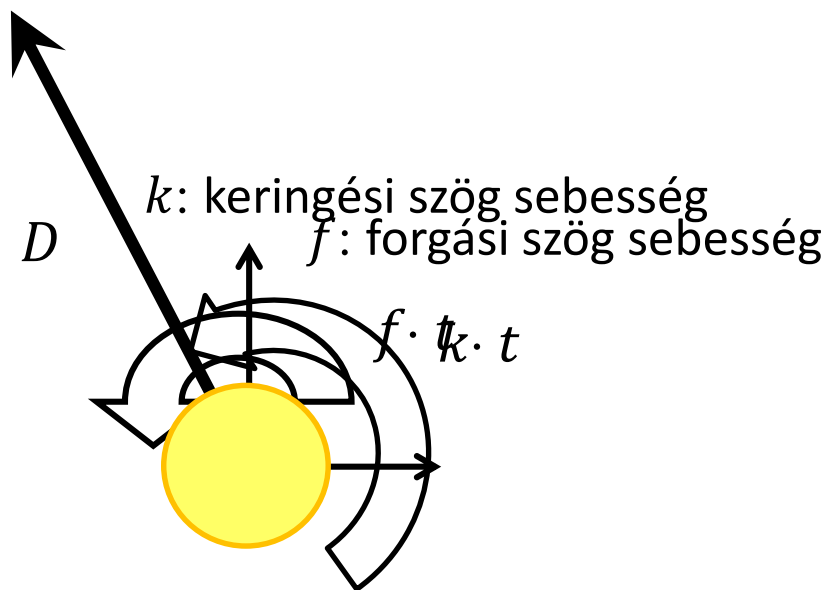


Animate: A Föld forog és kering a Nap körül

$$M = \begin{bmatrix} \cos(f \cdot t) & \sin(f \cdot t) & 0 & 0 \\ -\sin(f \cdot t) & \cos(f \cdot t) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(23^\circ) & \sin(23^\circ) & 0 \\ 0 & -\sin(23^\circ) & \cos(23^\circ) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ D \cos(k \cdot t) & D \sin(k \cdot t) & 0 & 1 \end{bmatrix}$$



Planet

```
const vec3 X(1,0,0), Z(0,0,1);

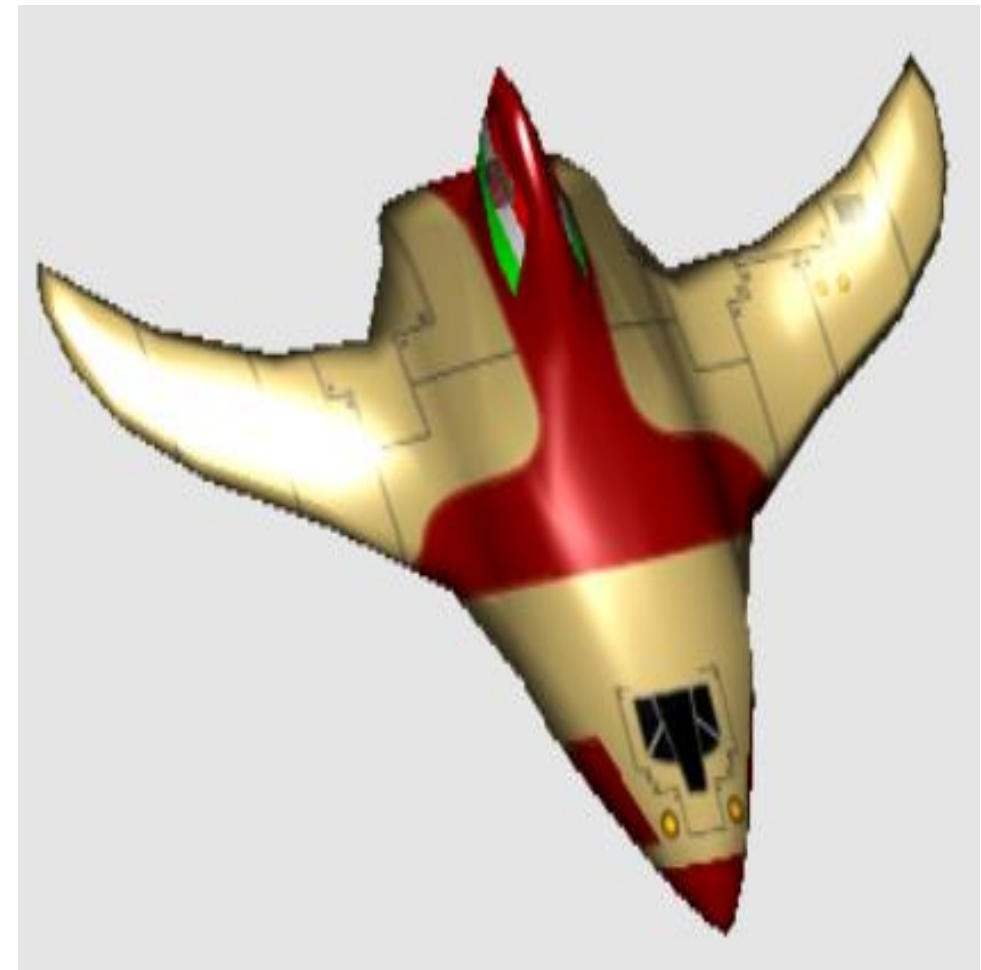
class Planet : public GameObject {
    float rotAng = 0, revAng = 0; // animation state
    float rotVel, revVel, tilt, D; // animation parameter
public:
    Planet(float _rotVel, float _revVel, float _tilt, float _D) { ... }

    void Animate(float dt) {
        rotAng += rotVel * dt;
        revAng += revVel * dt;
    }

    void ModelingTransform(RenderState& state) {
        vec3 p = vec3(cos(revAng), sin(revAng), 0) * D;
        state.M = RotationMatrix(rotAng, Z) * RotationMatrix(tilt, X) *
            TranslateMatrix(p) * state.M;
        state.Minv = state.Minv * TranslateMatrix(-p) * RotationMatrix(-tilt,X) *
            RotationMatrix(-rotAng,Z);
    }
};
```

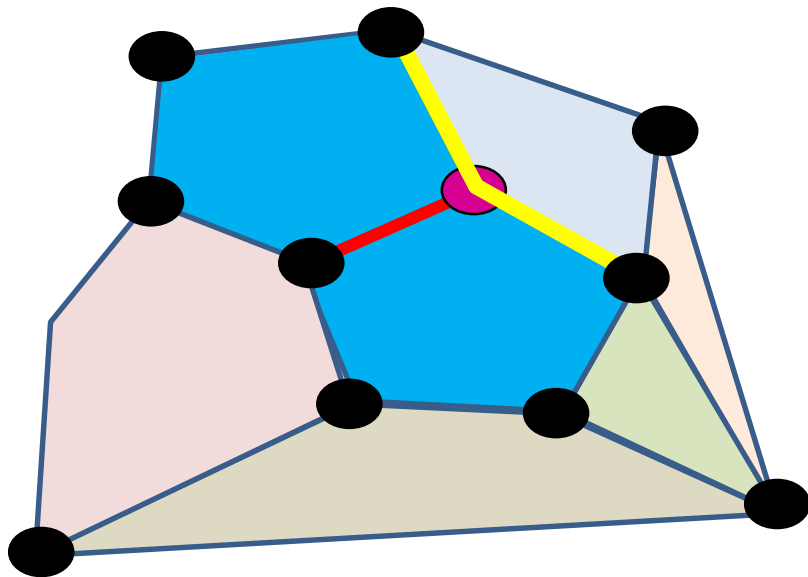
Az űrhajó

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



Az Euler karakterisztika invariáns: csúcs - él + lap = χ

χ csak felület topológiájától függ, amelyre a csúcsokat, éleket és lapokat felrajzoltuk.



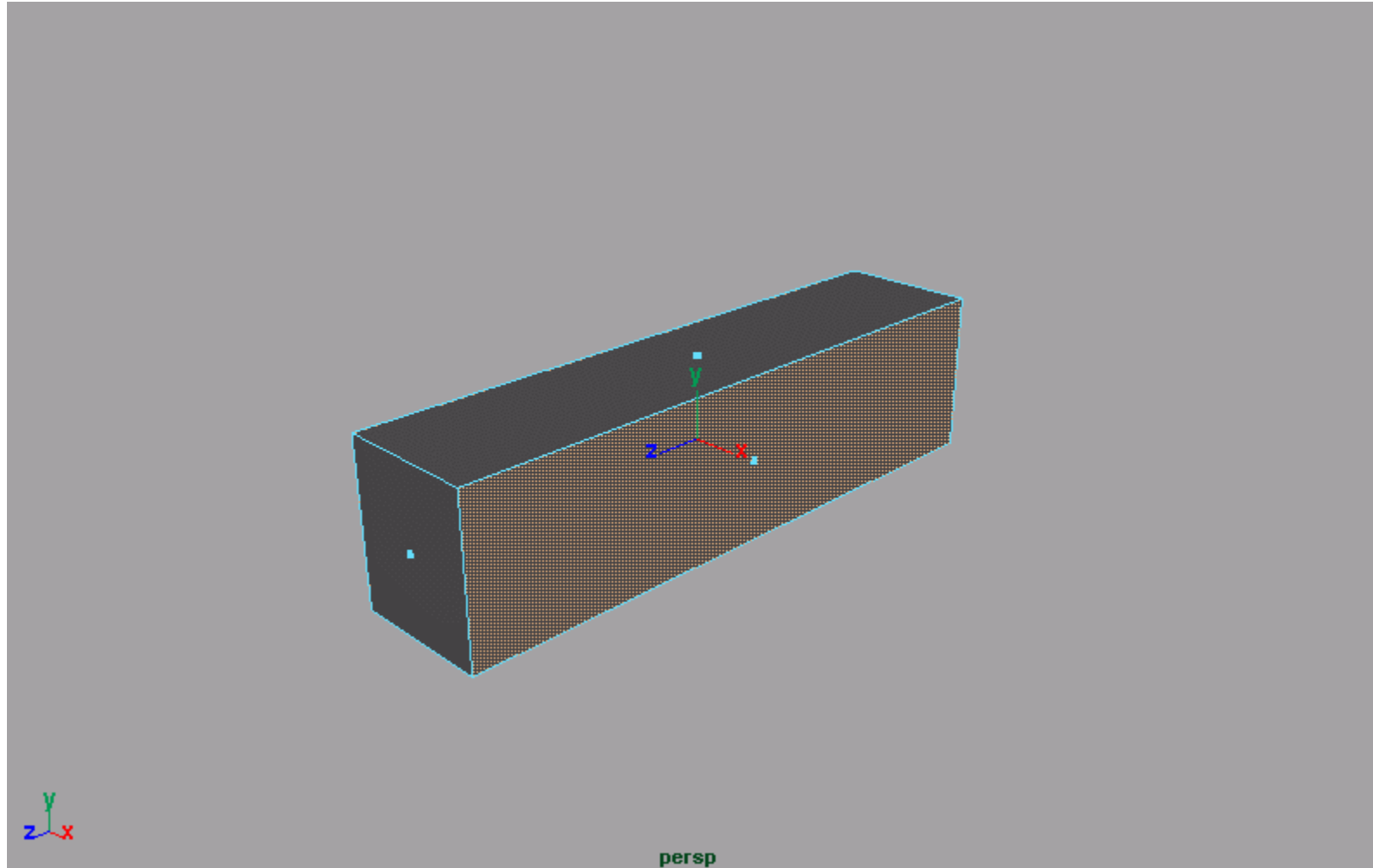
csúcs - él + lap

+1 -1

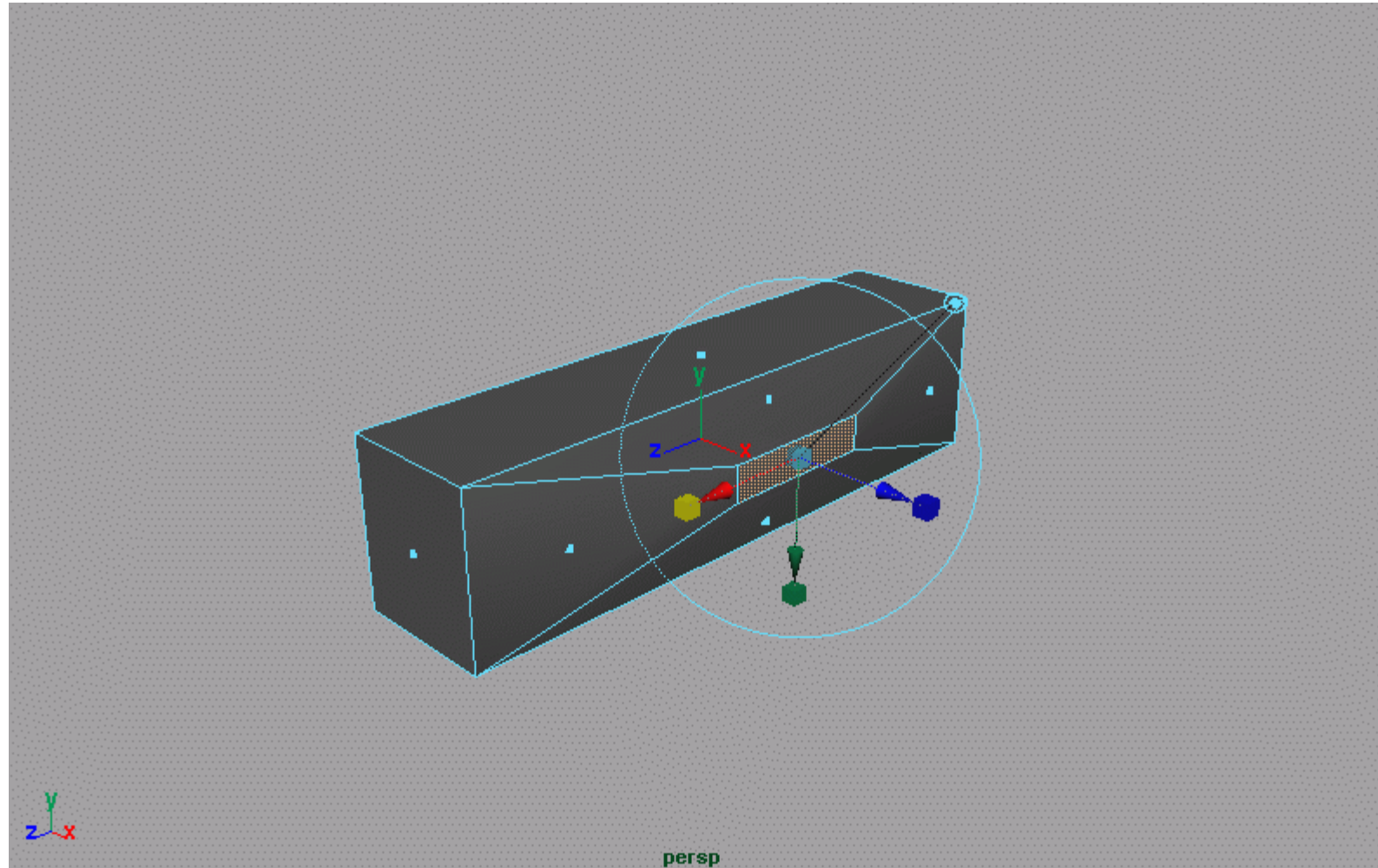
-1 +1

Úrhajó geometria

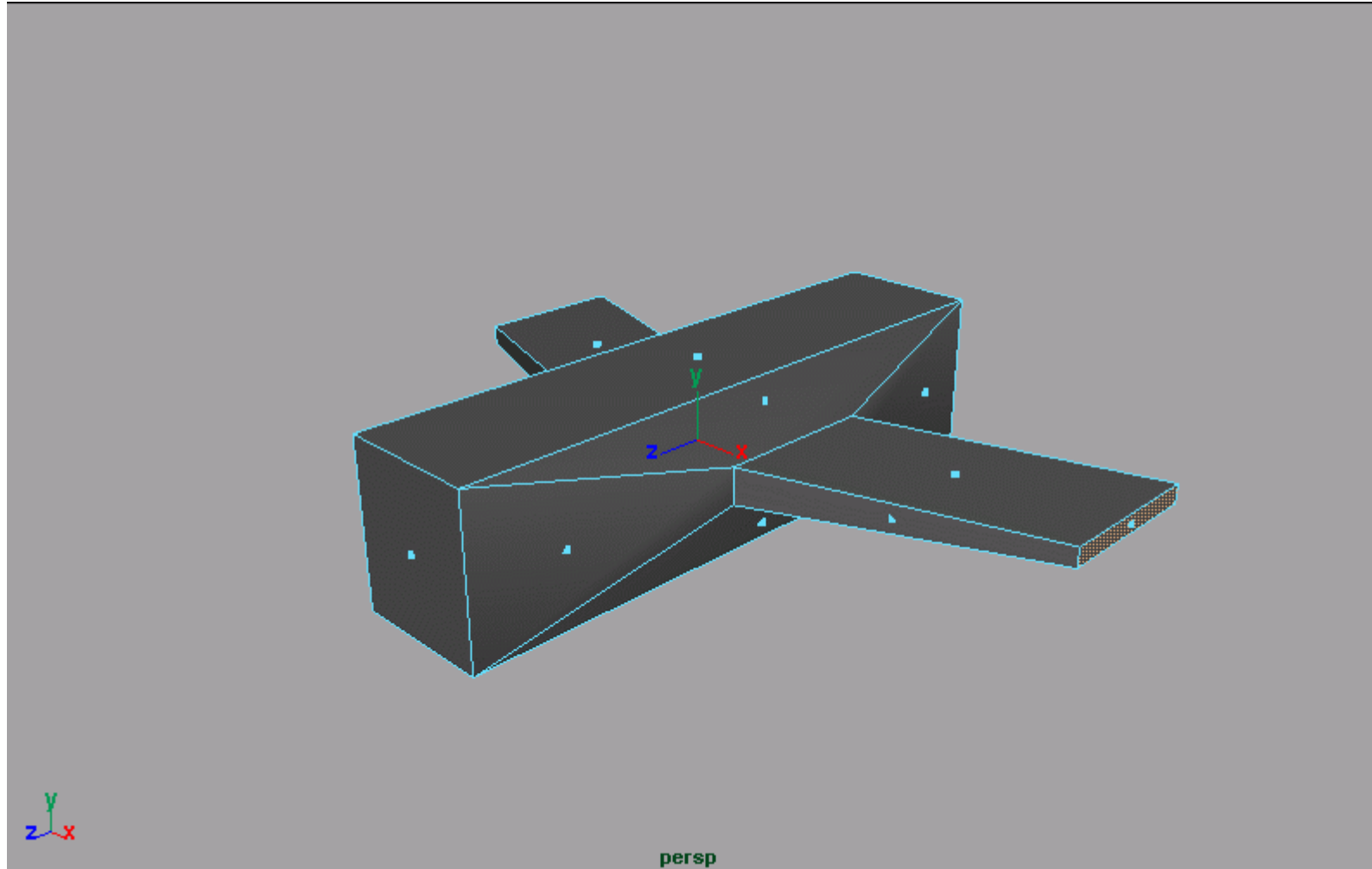
Euler műveletek: csúcs + lap = él + 2



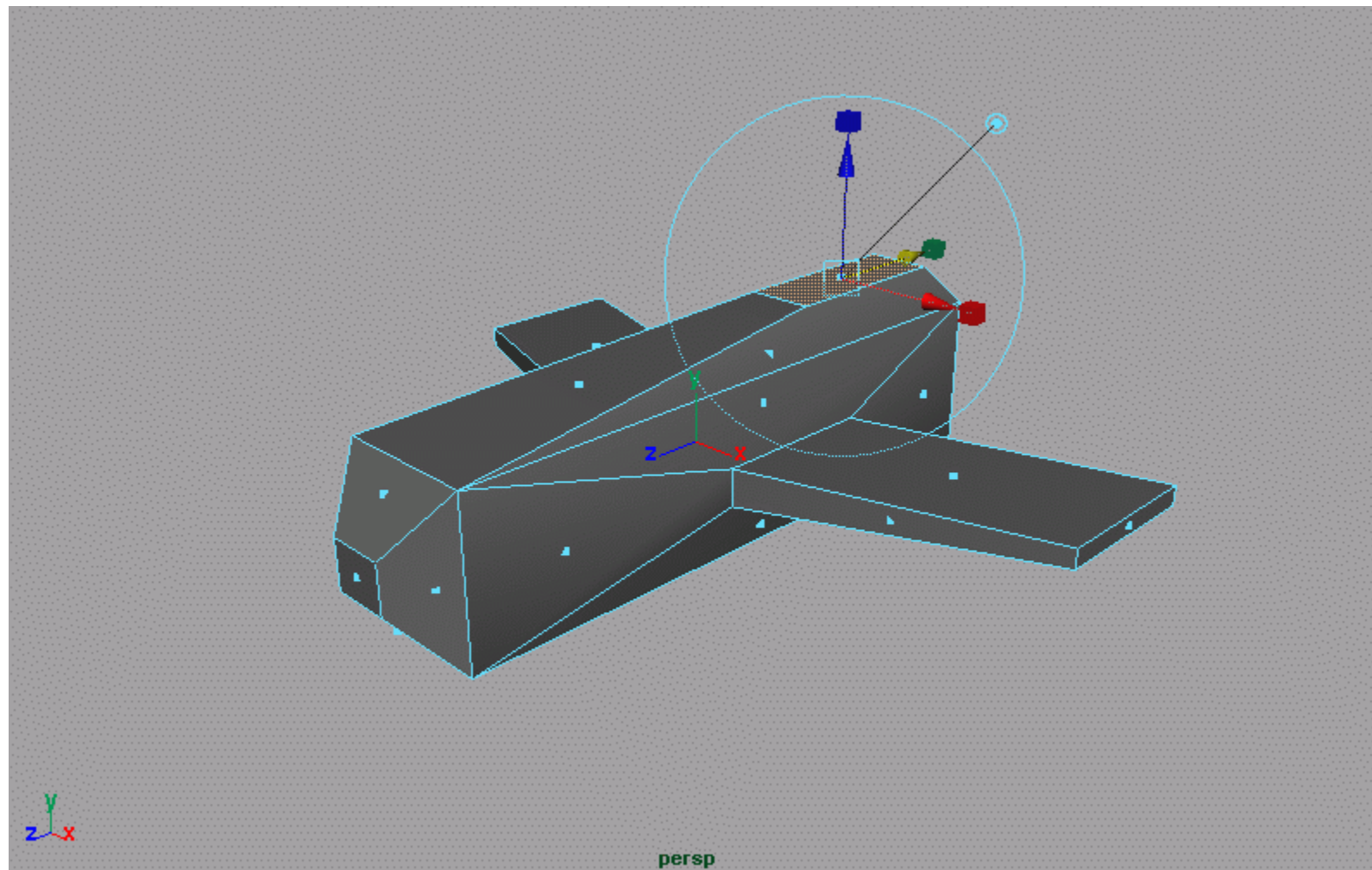
Úrhajó geometria



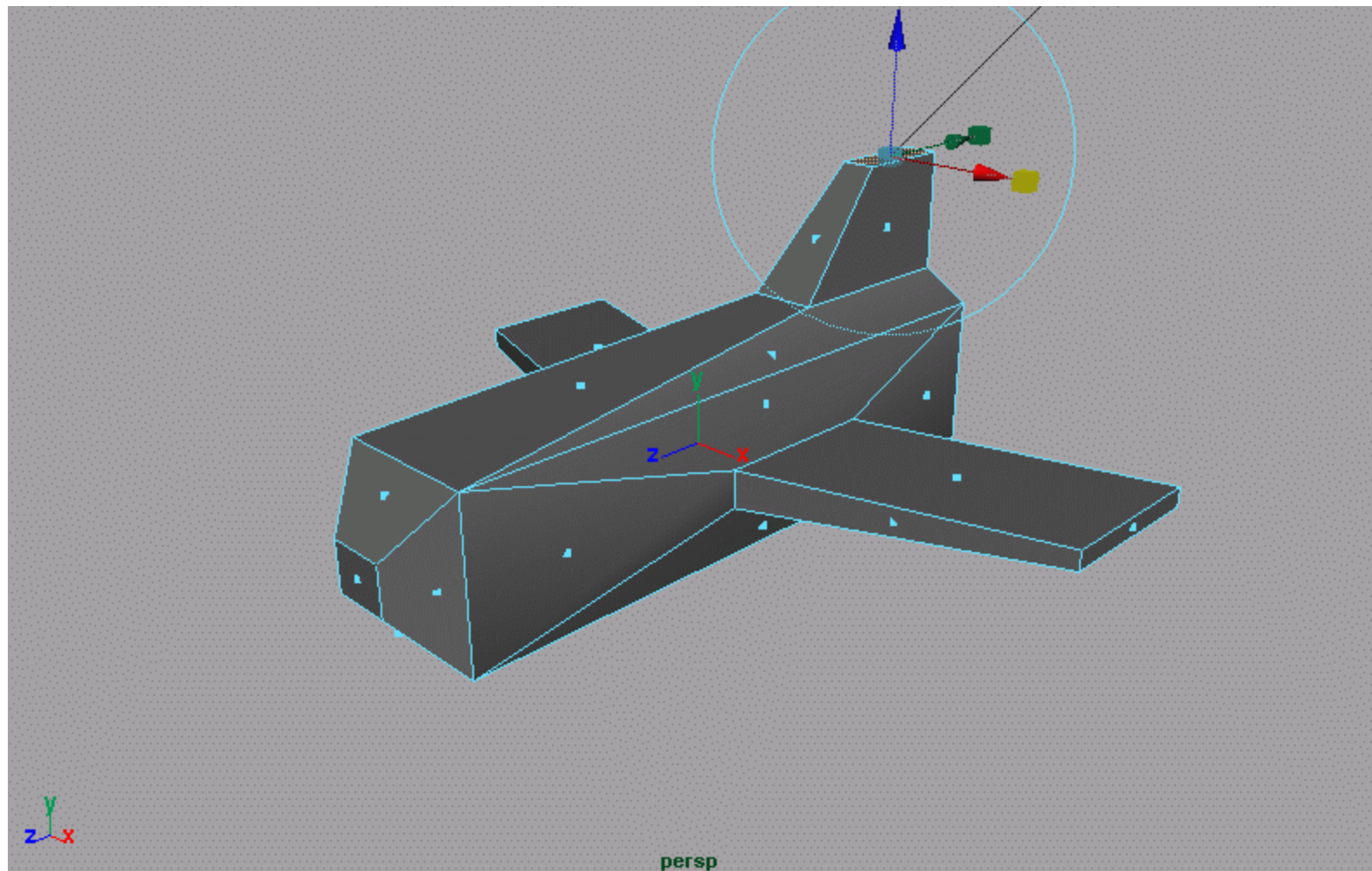
Úrhajó geometria



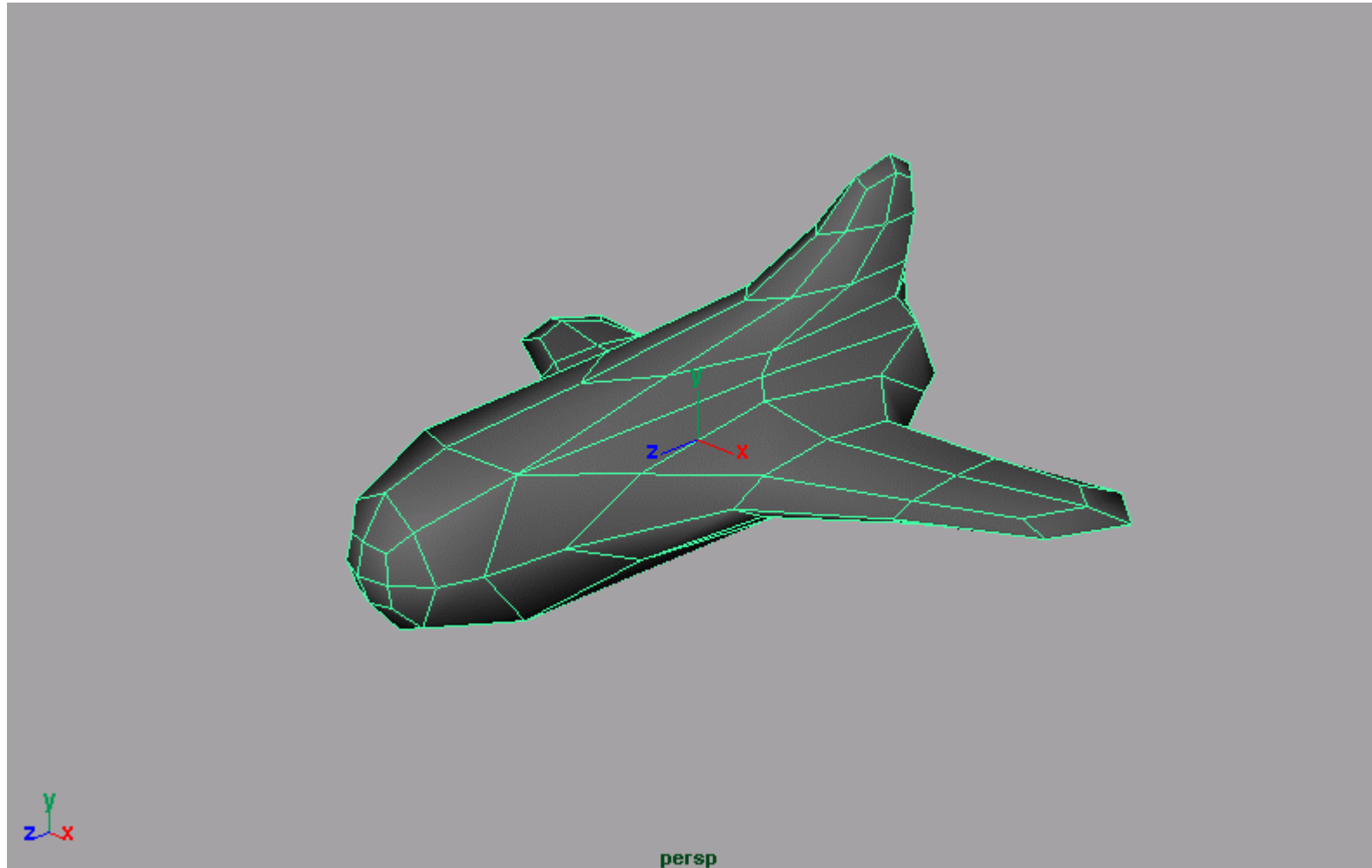
Úrhajó geometria



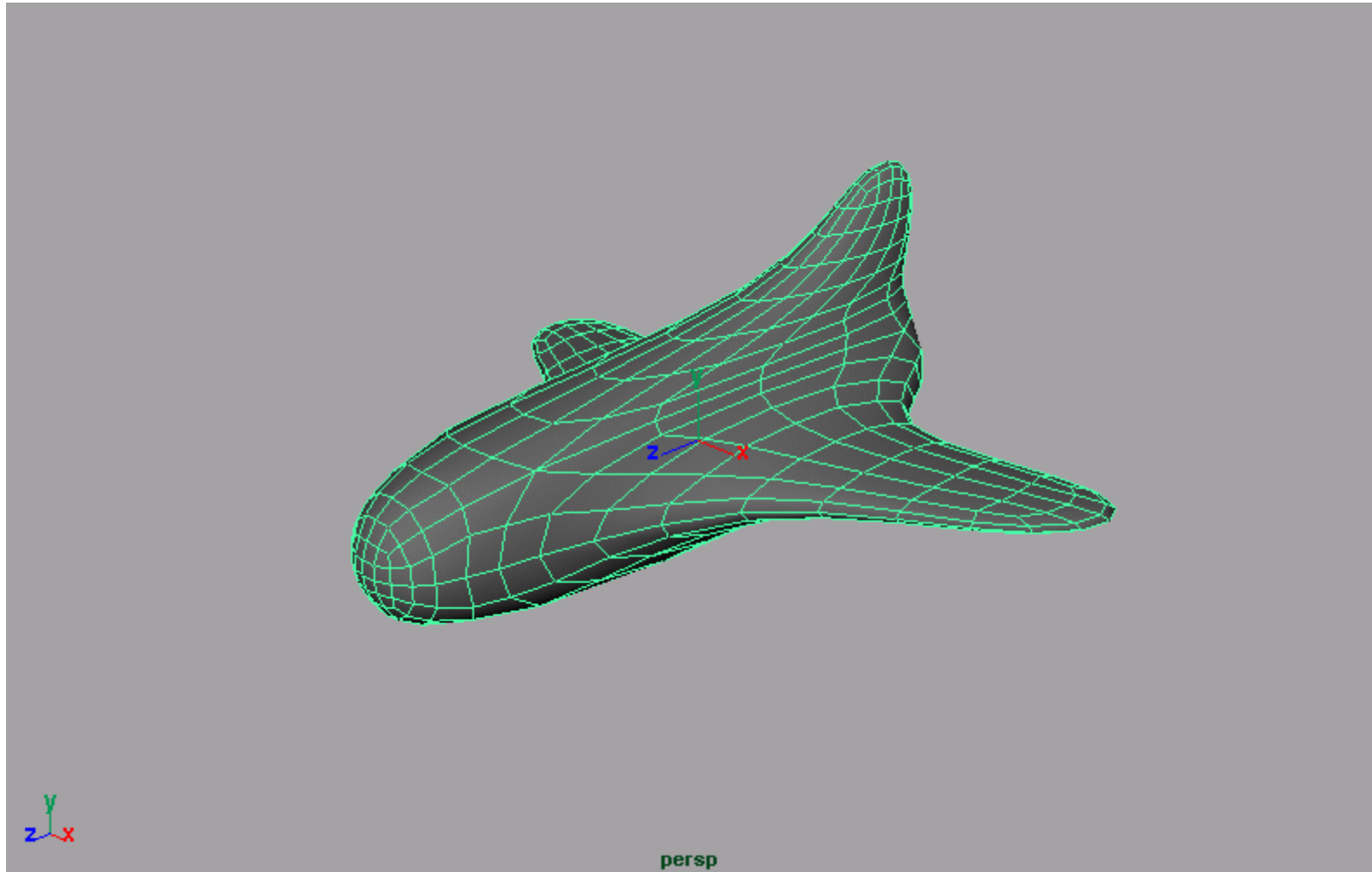
Úrhajó geometria



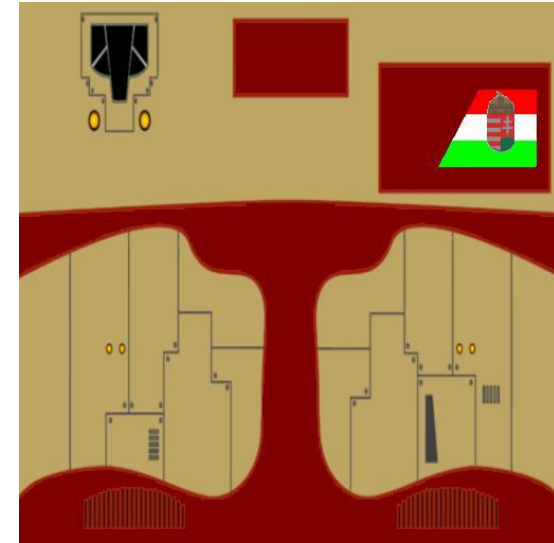
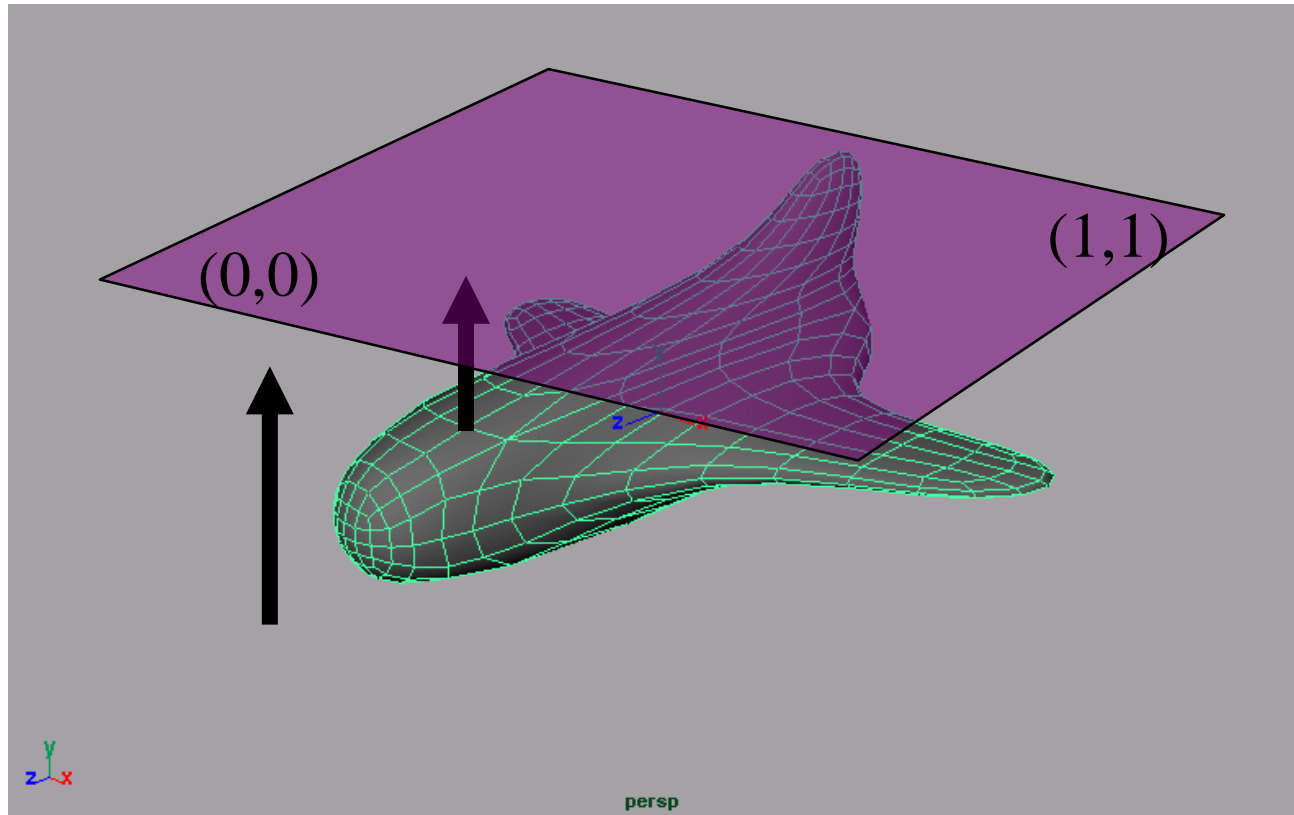
Úrhajó geometria: Catmull-Clark subdivision



Úrhajó geometria: Catmull-Clark subdivision



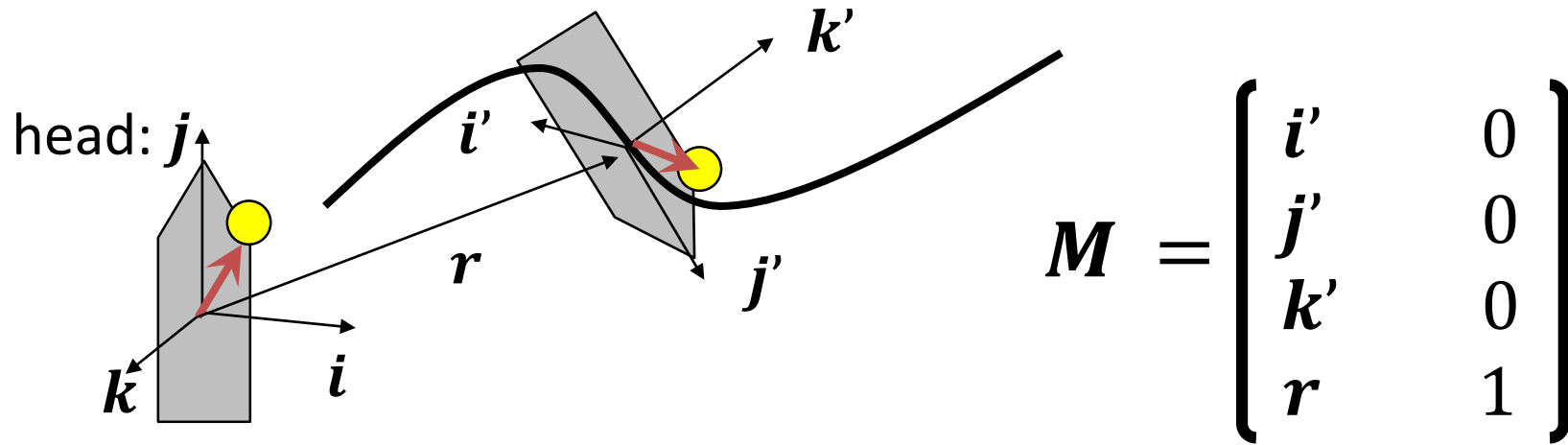
Textúrához paraméterezés



OBJ fájlformátum

```
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: Frenet frame + Newton 2



Dinamikai szimuláció:

$$F = ma, \quad a = \frac{dv}{dt}, \quad v = \frac{dr}{dt} \quad \Rightarrow \quad a = \frac{F}{m}, \quad v += a\Delta t, \quad r += v\Delta t$$

Orientáció

$$j' = v$$

$$k' = a$$

$$i' = j' \times k'$$

Orientáció, nem ortonormált:

$$j' = v$$

$$k^* = k'(1 - \alpha) + a\alpha$$

$$i' = j' \times k^*$$

Gram-Schmidt ortogonalizáció:

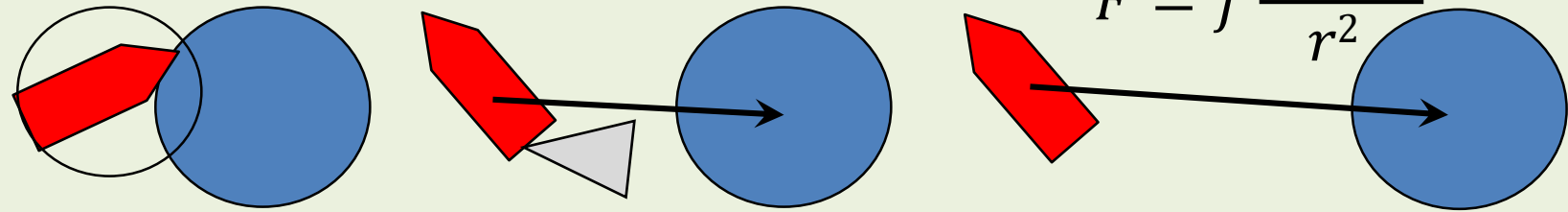
$$j' = v/|v|$$

$$i' = j' \times k^*/|j' \times k^*|$$

$$k' = i' \times j'$$

Ship :: Control

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

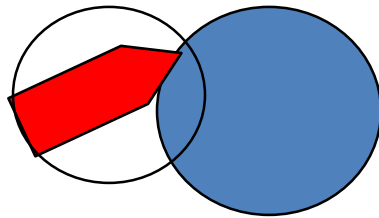


```
    }  
    if (dynamic_cast<Avatar*>(obj)) {
```



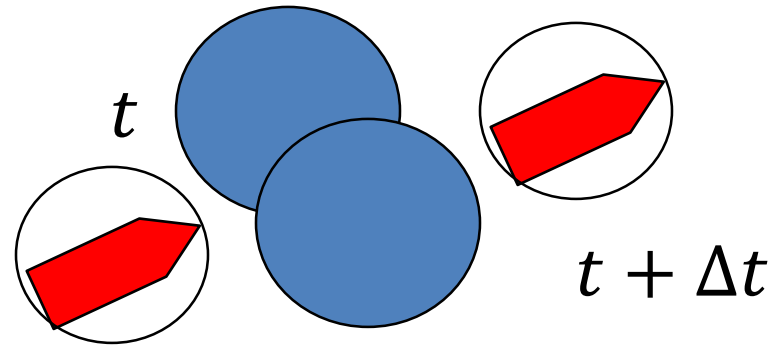
```
    }  
}
```

Ütközésetektálás: lassú objektumok



adott t

Probléma, ha az objektum gyors

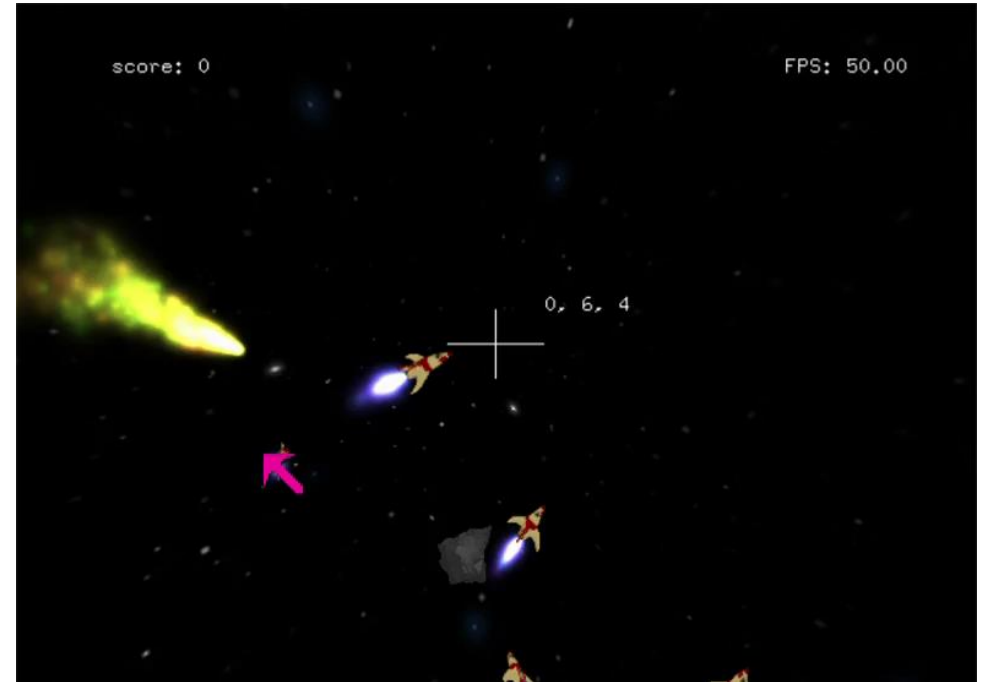


```
dist = length(obj1.pos - obj2.pos)
minDist = obj1.BoundingRadius() + obj2.BoundingRadius()
if (dist < minDist) Collision!
```

Foton torpedó

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

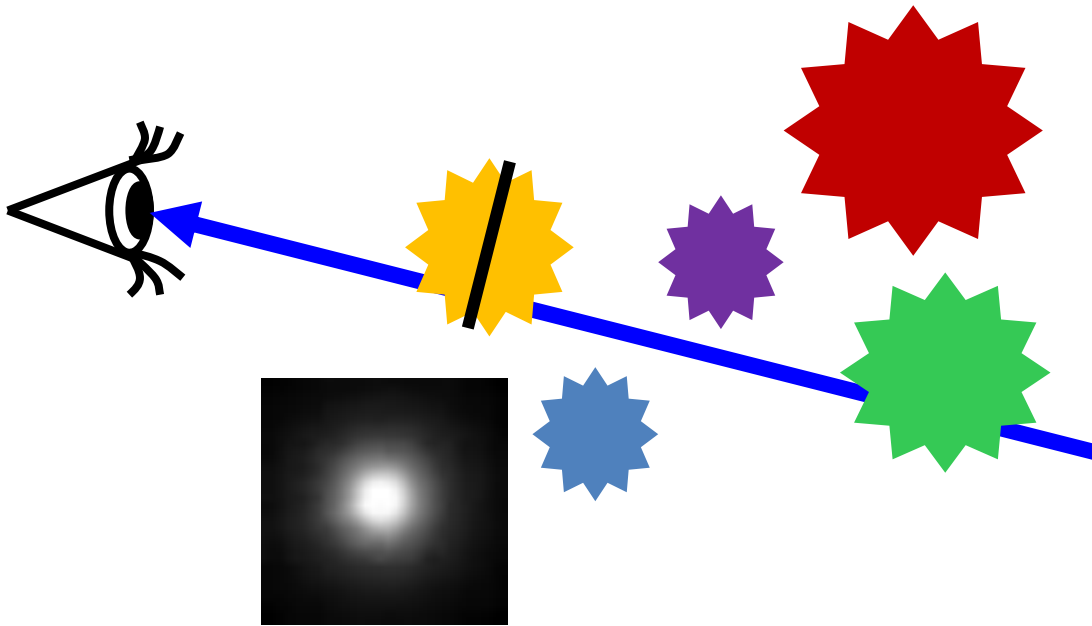
átlátszó



- Ütközésetektálás = gyors mozgás

Billboard

Egyetlen félig átlátszó textúra egy négyzeten



- Ponton változó textúrankoordináták?
- Pont méret: `glPointSize` VAO-nként egyszer?
- Átlátszóság?



Volumetric shader

Vertex shader:

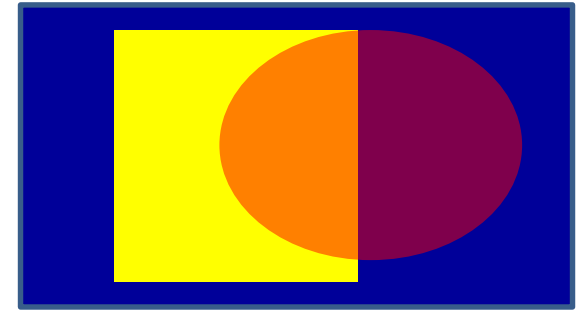
```
uniform mat4 VP;
uniform int vpHeight;
layout(location = 0) in vec3 wPos;
layout(location = 1) in vec4 color;
layout(location = 2) in float size;
out vec4 col;

void main() {
    gl_Position = vec4(wPos, 1.0f) * VP;
    gl_PointSize = size * vpHeight / gl_Position.w ;
    col = color;
}
```

Pixel shader:

```
uniform sampler2D texMap;
in vec4 col;
out vec4 fragColor;

void main() {
    fragColor = texture(texMap, gl_PointCoord) * col;
}
```

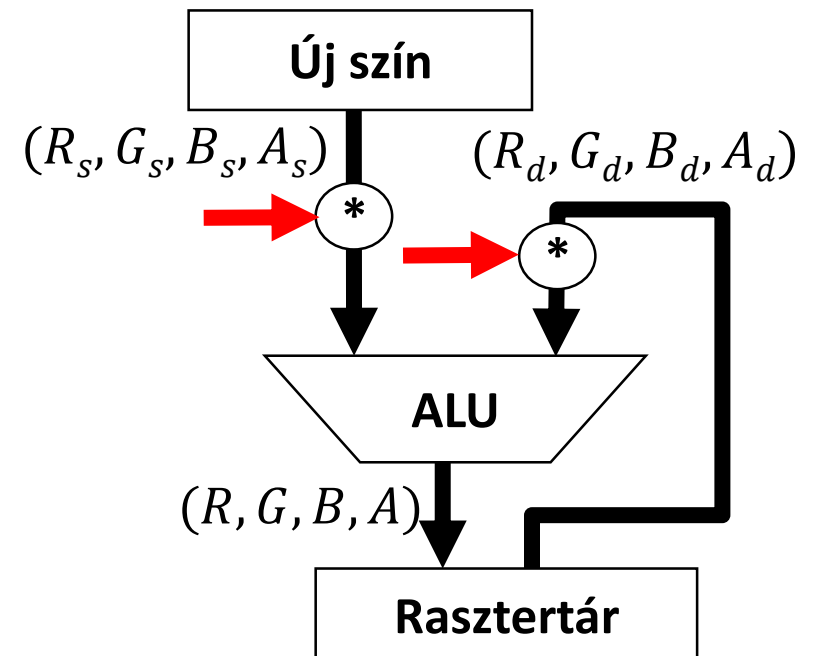


CPU program:

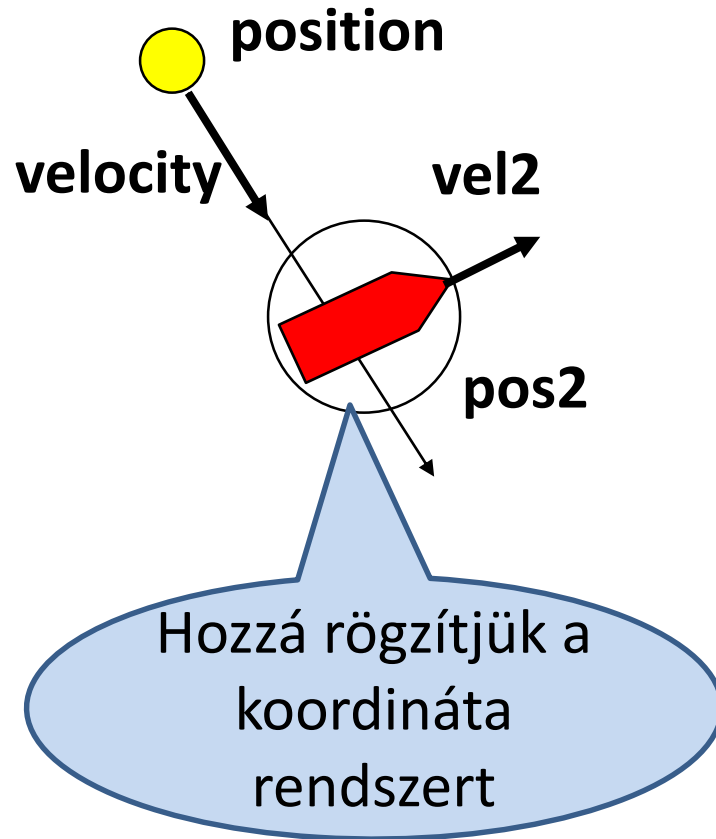
```
glEnable(GL_PROGRAM_POINT_SIZE);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE);
```

VBO feltöltés a világkoord-ban

Draw



Gyors (folytonos) ütközés detektálás



$$\text{rel_pos} = \text{position} - \text{pos2}$$

$$\text{rel_velocity} = \text{velocity} - \text{vel2}$$

$$\text{Ray: } \text{rel_pos} + \text{rel_velocity} \cdot t$$

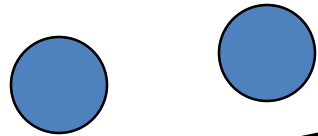
If (ray intersects bounding sphere first
&& tintersect < dt) **Collision!**

Robbanás

- Nagyon komplex geometria
- Hasonló kinézet minden irányból
- Plakátgyűjtemény
- Részecske rendszer

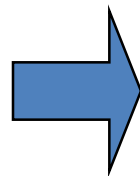


Részecske rendszerek



Globális erőter
(szél fújja a füstöt)

Véletlen
Kezdeti
értékek



pos:

velocity:

acceleration:

lifetime

age:

size, dsize:

weight, dweight:

color, dcolor:

$pos += velocity * dt$

$velocity += acceleration * dt$

$acceleration = force / weight$

$age += dt; \text{if } (age > lifetime) \text{ Kill}();$

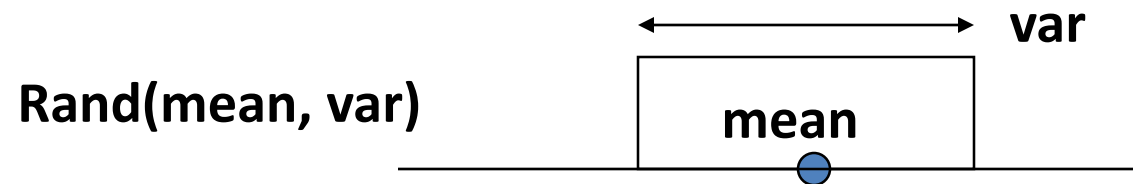
$size += dsize * dt;$

$weight += dweight * dt$

$color += dcolor * dt$



Robbanás paraméterei



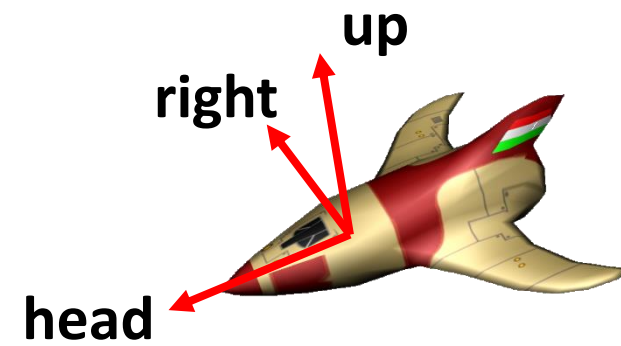
```
pos = center; // kezdetben fókuszált
lifetime = Rand(2, 1);

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;
```

Player



```
class Player : public Avatar {  
    void ProcInput() {  
        if ( isKeyPressed( ` ` ) ) // Fire!  
            objects.push_back(new Bullet(pos, velocity));  
  
        // Kormányzás: az avatár koordinátarendszerében!  
        vec3 head = normalize(velocity);  
        vec3 right = normalize(cross(wVup(), head));  
        vec3 up = cross(head, right);  
  
        if (isKeyPressed(KEY_UP))      force -= up;  
        if (isKeyPressed(KEY_DOWN))    force += up;  
        if (isKeyPressed(KEY_LEFT))    force -= right;  
        if (isKeyPressed(KEY_RIGHT))   force += right;  
    }  
};
```