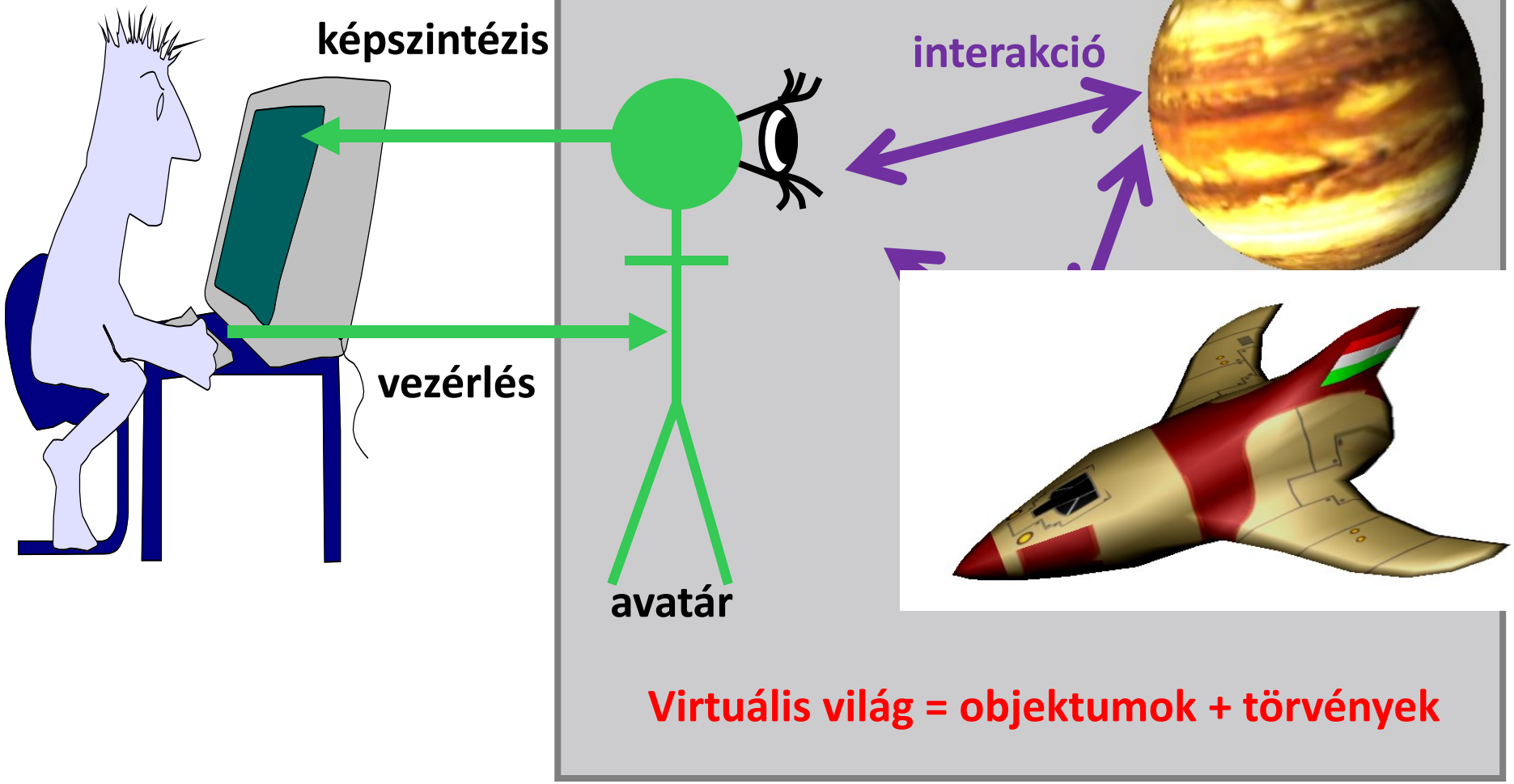
“The only legitimate use of a computer is to play games.”

Eugene Jarvis

Játékfejlesztés

Szirmay-Kalos László

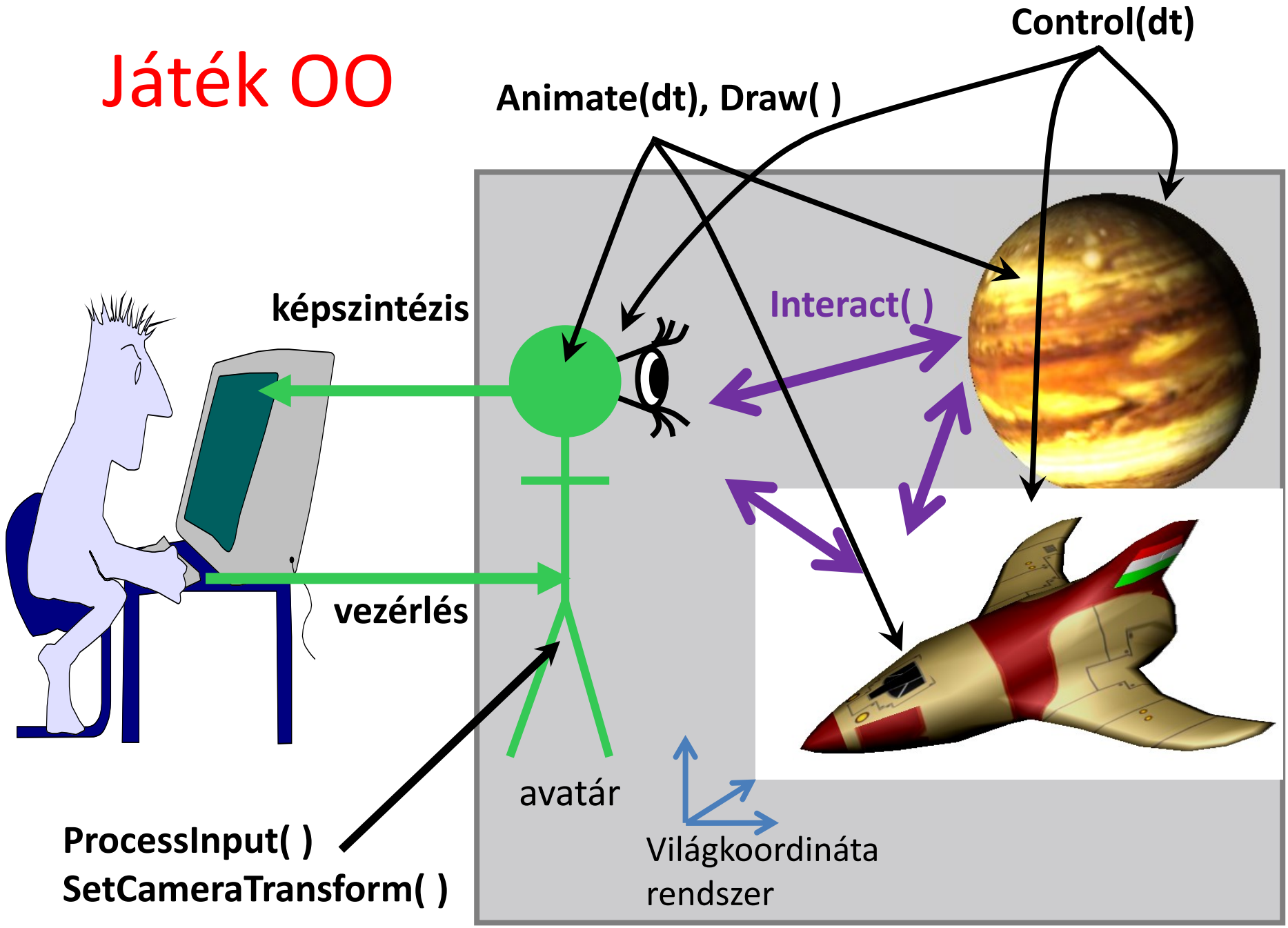
Virtuális valóság



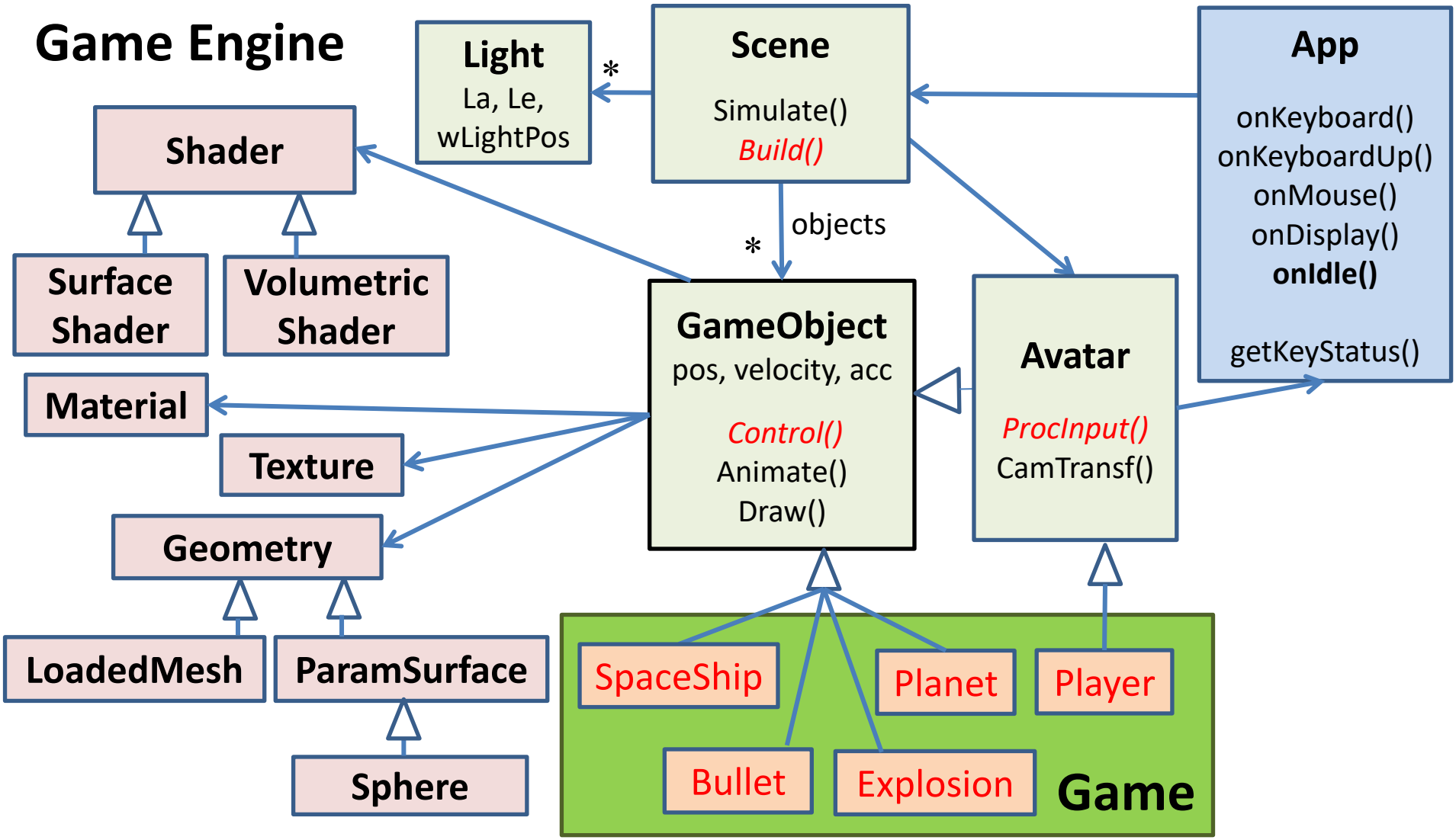
Játékok feladatai

- 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( )
Világkoordináta
rendszer

Osztálydiagram



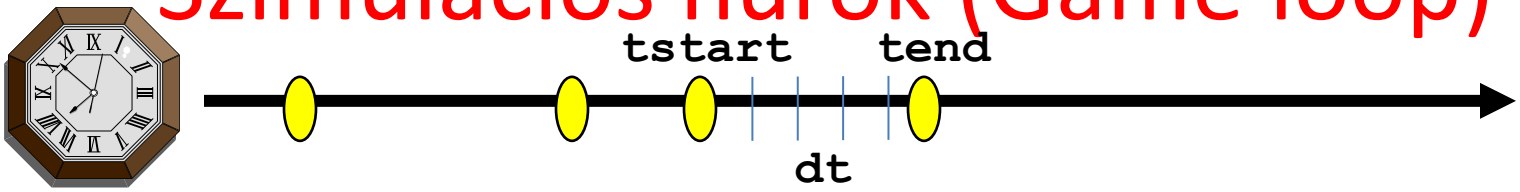
Játékobjektum (GameObject)

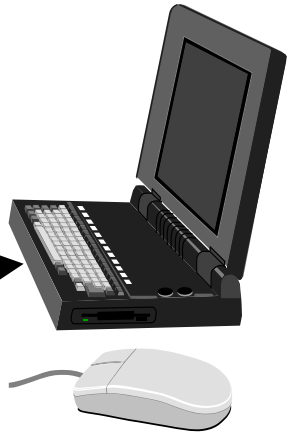
```
class GameObject {
protected:
   Shader *   shader;
   Material * material;
   Texture *  texture;
   Geometry * geometry;
   vec3 pos, velocity, acceleration;
public:
   GameObject(Shader* s, Material* m,
              Texture* t, Geometry* g) { … }
   virtual void Control(float dt) { }
   virtual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

Virtuális világ

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

Szimulációs hurok (Game loop)

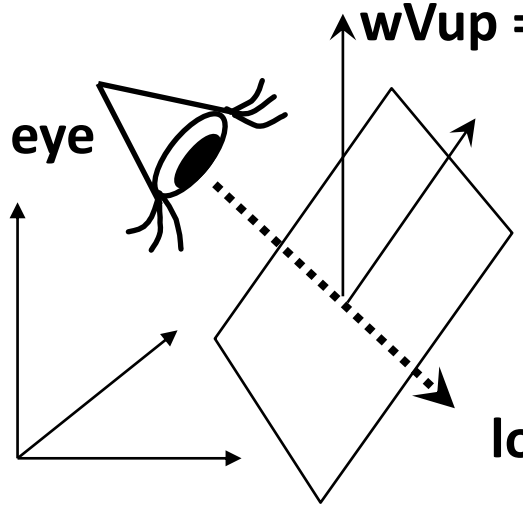




```
void onIdle ( ) {    // idle call back
    static float tend = 0;
    float tstart = tend;
    tend = glutGet(GLUT_ELAPSED_TIME)/1000.0f;
    avatar->ProcessInput( );
    for(float t = tstart; t < tend; t += dt) {
        float Dt = min(dt, tend - t);
        for (GameObject * obj : objects) obj->Control(Dt);
        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( );
}
```

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();
    }
};
```



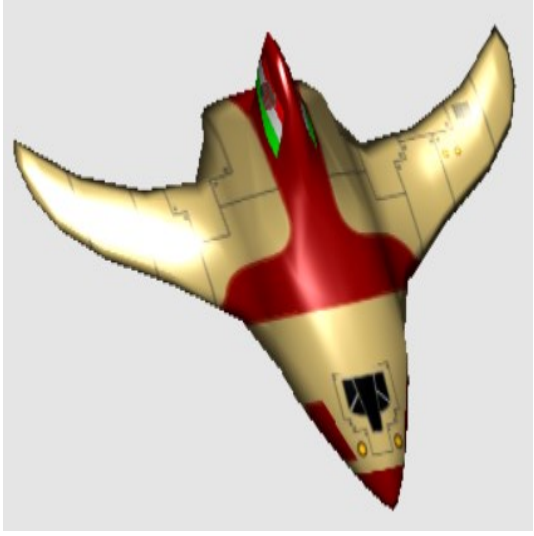
wVup = [0, 1, 0] vagy a
gyorsulásból és a korábbi wVup átlagából

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

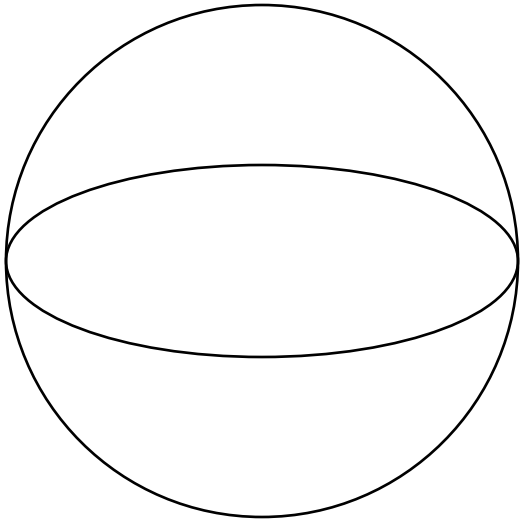
Madách Imre

Játékfejlesztés
2. Játékobjektumok

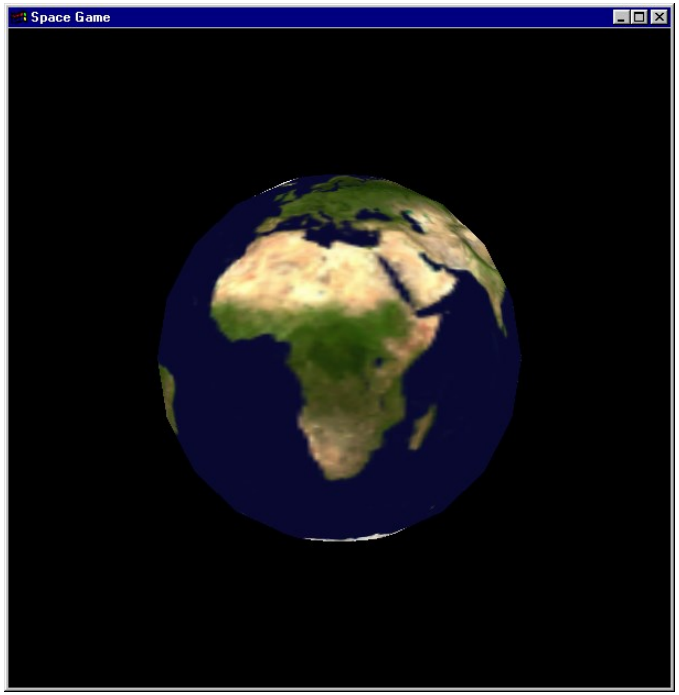
Szirmay-Kalos László



Bolygó: Planet



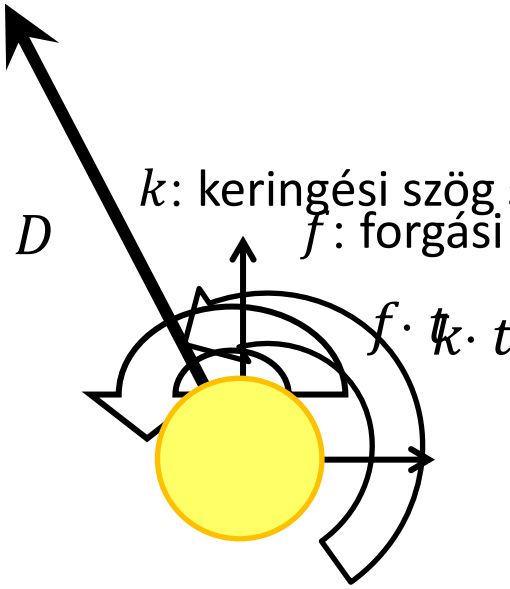
- Geometria: gömb
- Textúra
- Fizikai vagy
  - Tájékozódik majd követi a gravitációs törvényt
- <u>Képletanimáció:</u>
  - „beégetett pálya”
  - Többiek érdektelenek
  - Nincs respektált törvény



Animate:
A Föld forog és kering a Nap körül

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

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(23°) & \sin(23°) & 0 \\ 0 & -\sin(23°) & \cos(23°) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

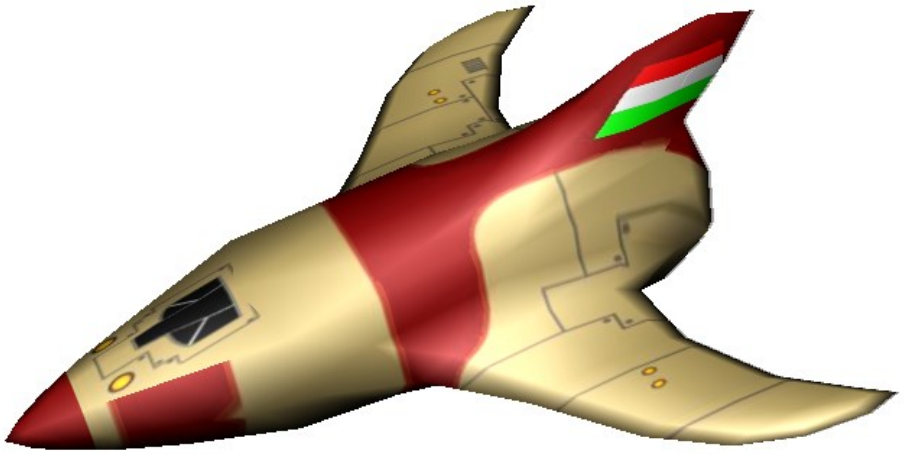


$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ D\cos(k \cdot t) & D\sin(k \cdot t) & 0 & 1 \end{bmatrix}$$

Planet

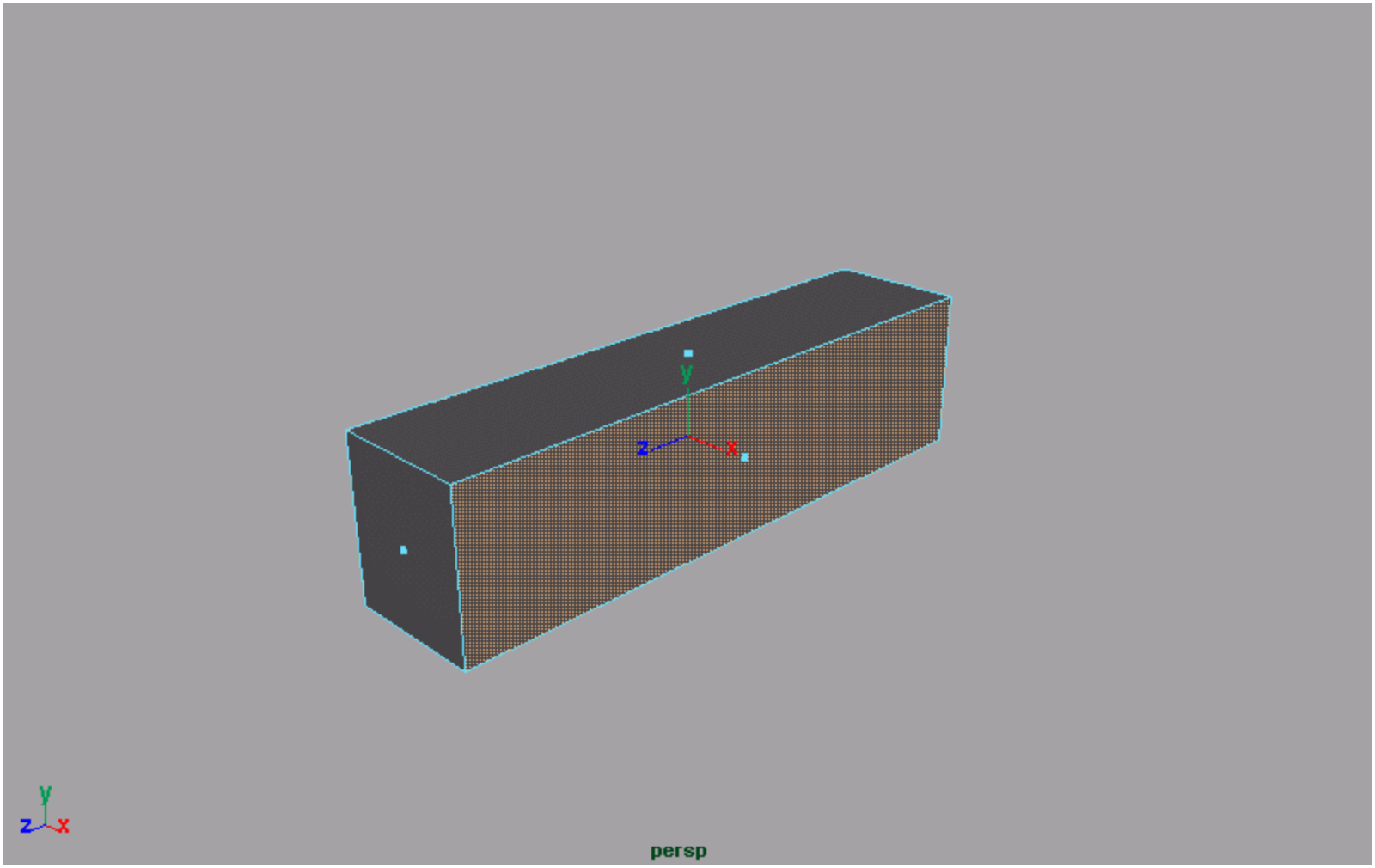
```
const vec3 X(1,0,0), Z(0,0,1);

class Planet : public GameObject {
   float rotAng = 0, revAng = 0;  // animation state
   float rotVel, revVel, tilt, D; // animation parameter
public:
   Planet(Shader* s, Material* m, Texture* t, Sphere* s,
          float rot, float rev, float ti, float d) :
       GameObject(s,m,t,s), rotVel(rot), revVel(rev), tilt(ti), D(d) {}

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

   void Draw(RenderState state) {
       vec3 p = vec3(cos(revAng), sin(revAng), 0) * D;
       state.M = RotationMatrix(rotAng, Z) * RotationMatrix(tilt, X) *
                 TranslateMatrix(p);
       state.Minv = TranslateMatrix(-p) * RotationMatrix(-tilt, X) *
                    RotationMatrix(-rotAng, Z);
       state.material = material; state.texture = texture;
       shader->Bind(state);
       geometry->Draw();
    }
};
```

Az űrhajó

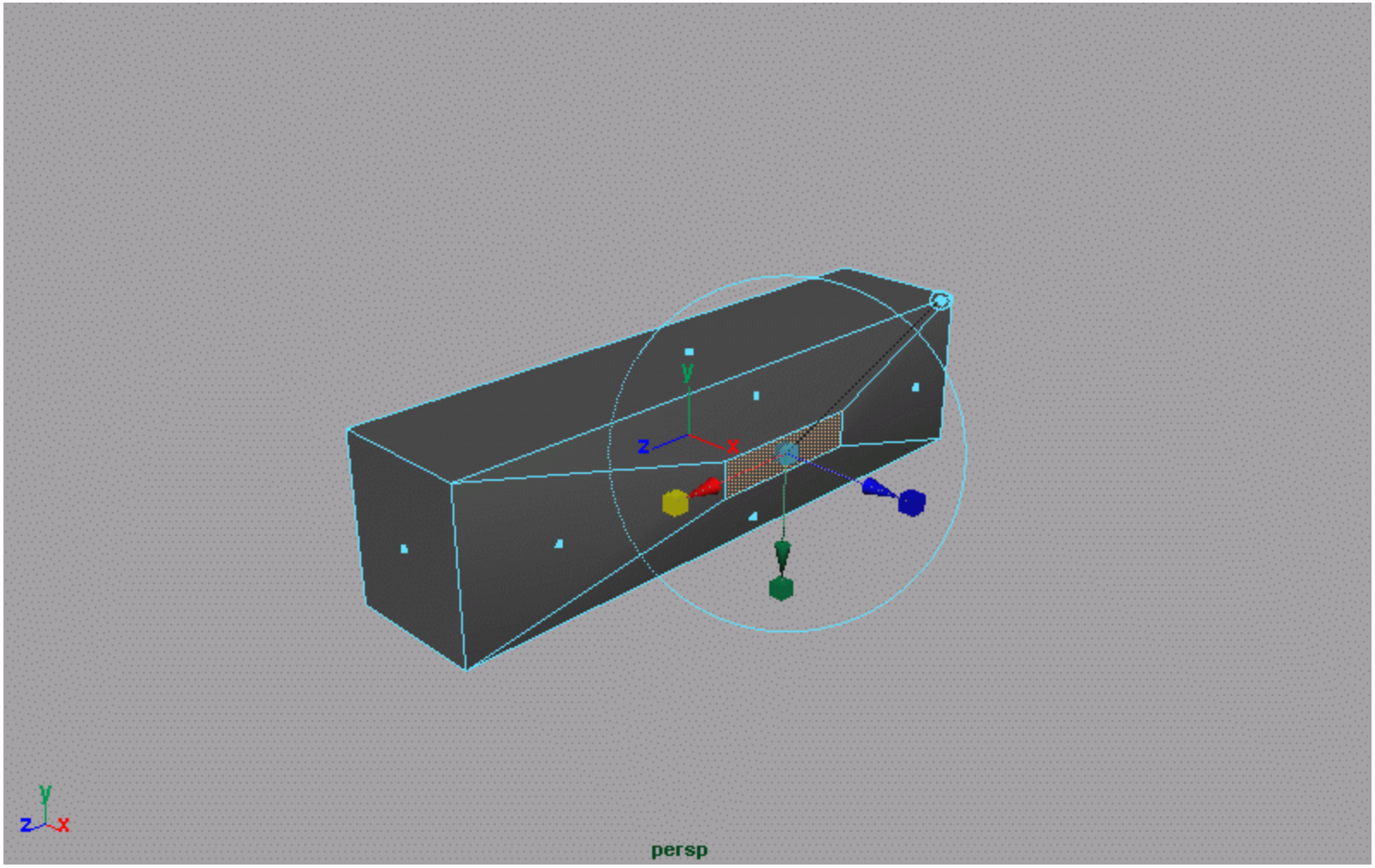


- Komplex geometria
  - négyszögháló
- Komplex textúra
- Fizikai animáció
  - erők (gravitáció, rakéták)
  - ütközések
- Viselkedés (AI)
  - A rakéták vezérlése
    - Ütközés elkerülés, avatártól menekülés, avatár üldözése

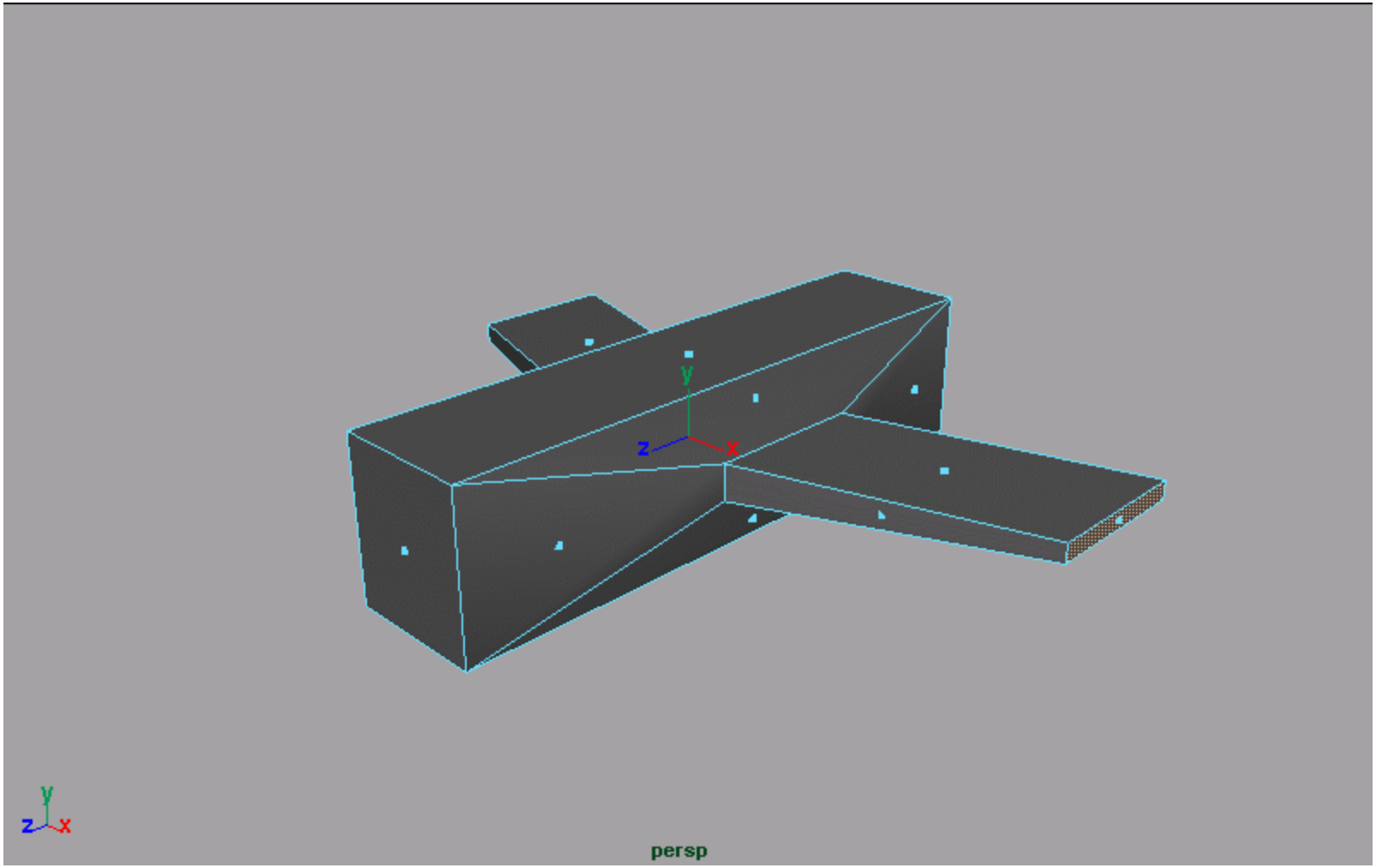
Űrhajó geometria
Euler műveletek: csúcs + lap = él + 2



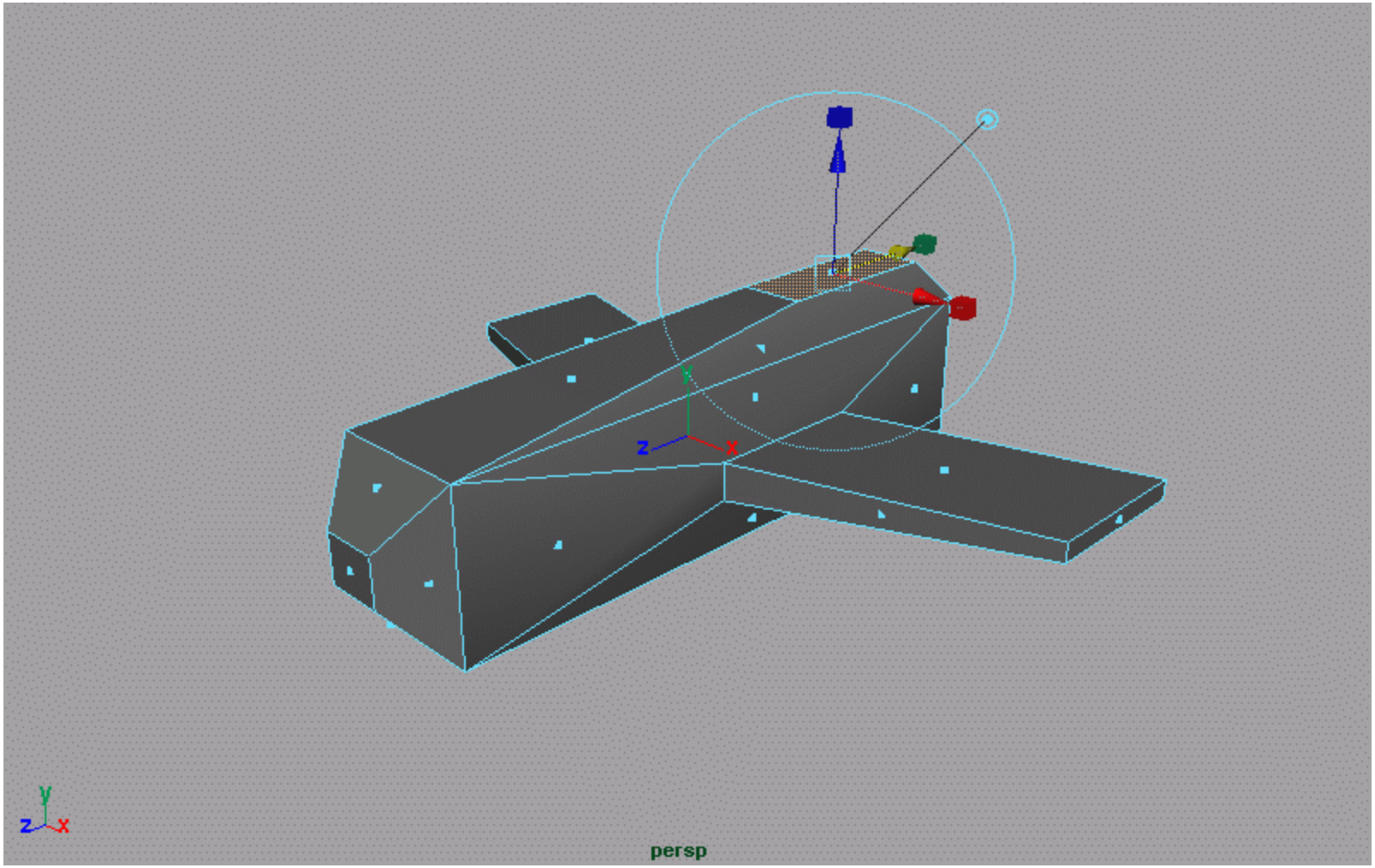
Űrhajó geometria



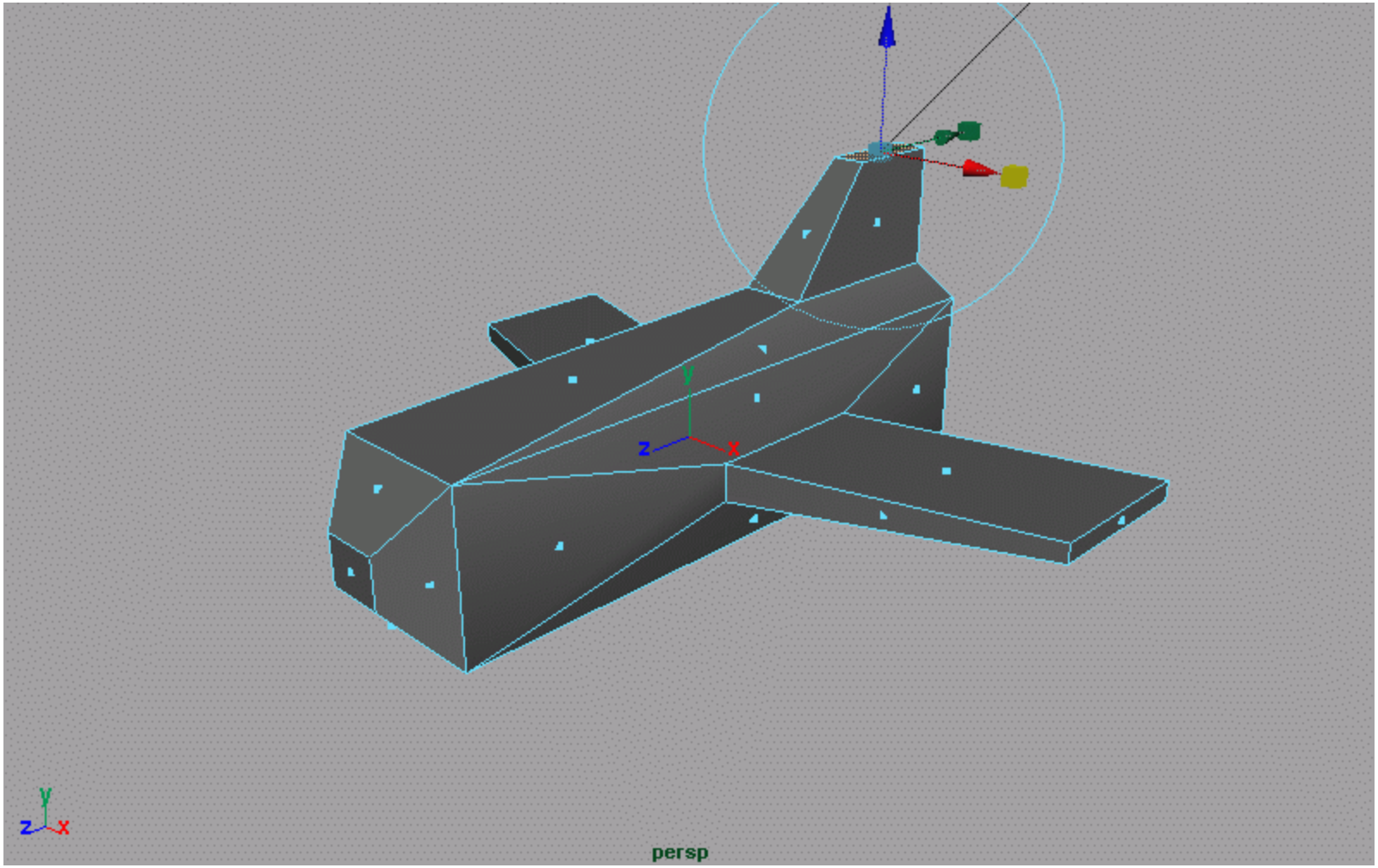
Űrhajó geometria



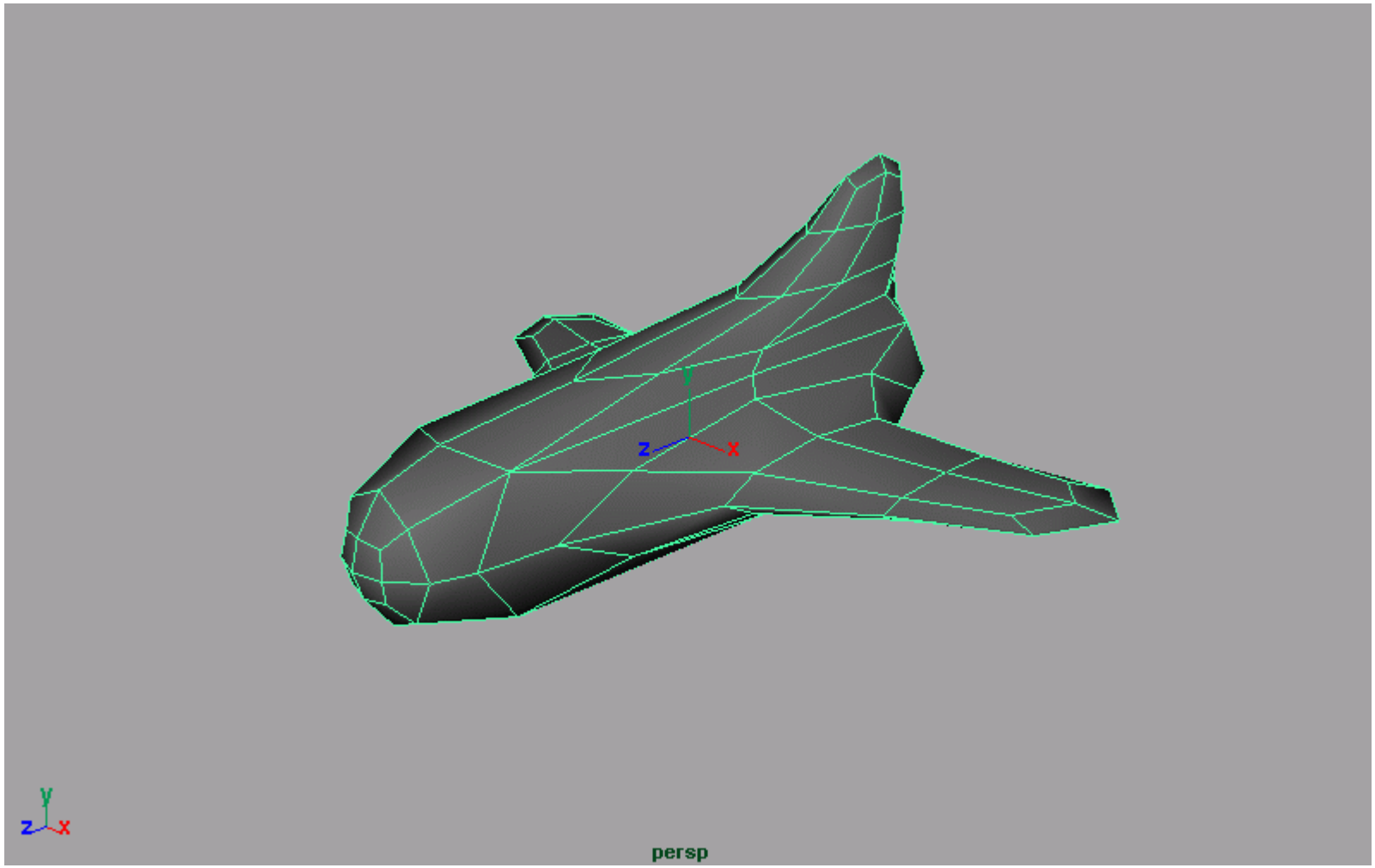
Űrhajó geometria



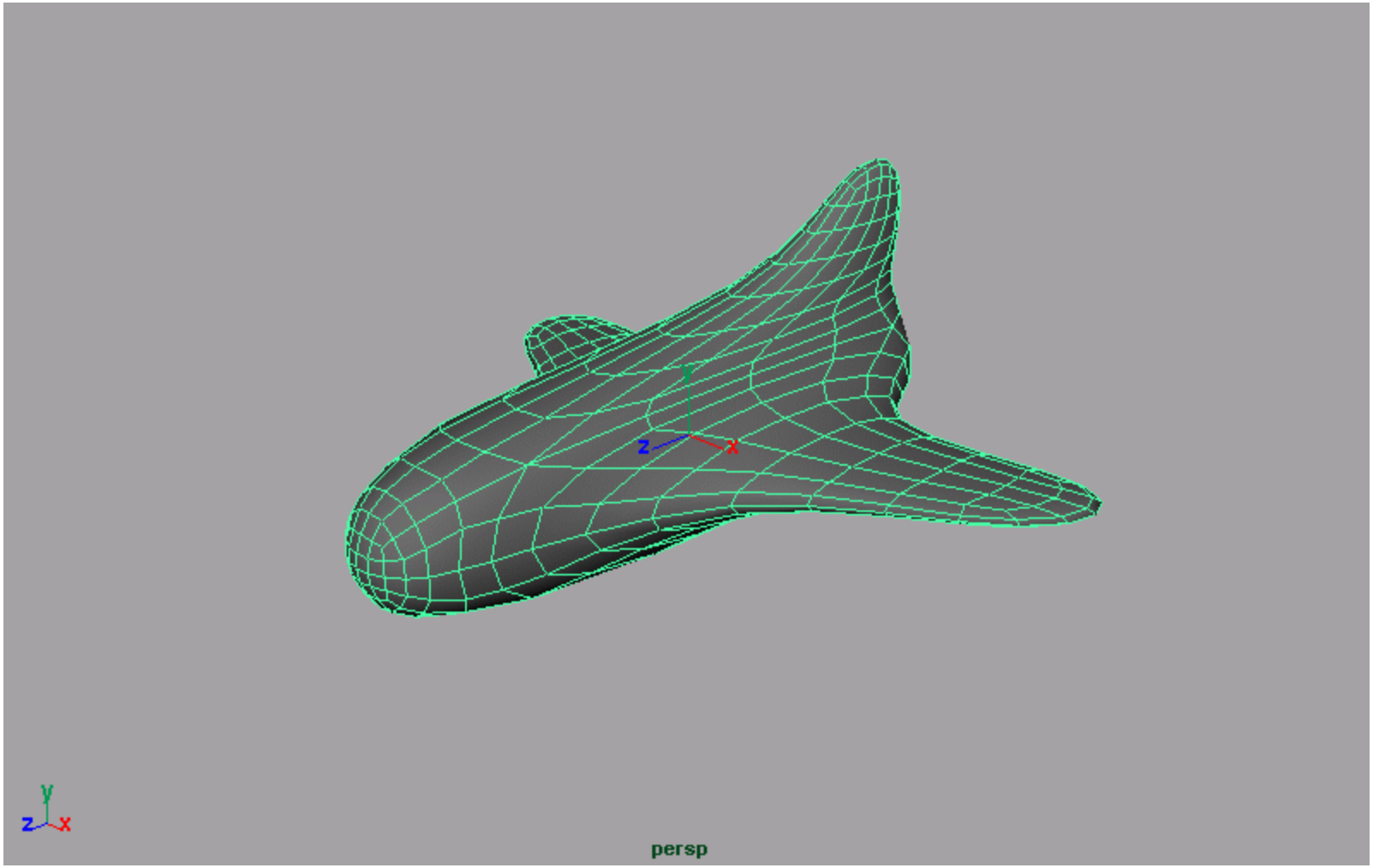
Űrhajó geometria



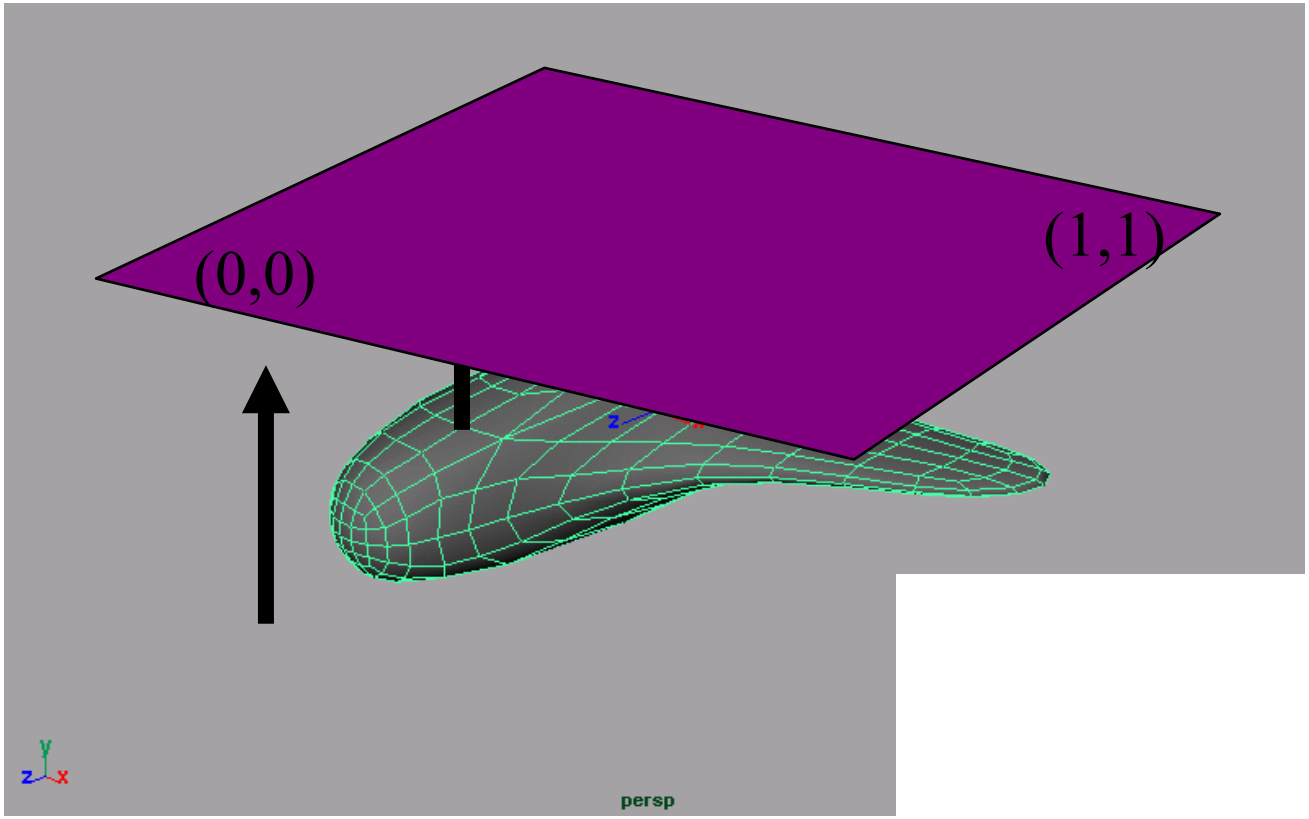
Űrhajó geometria: Catmull-Clark subdivision

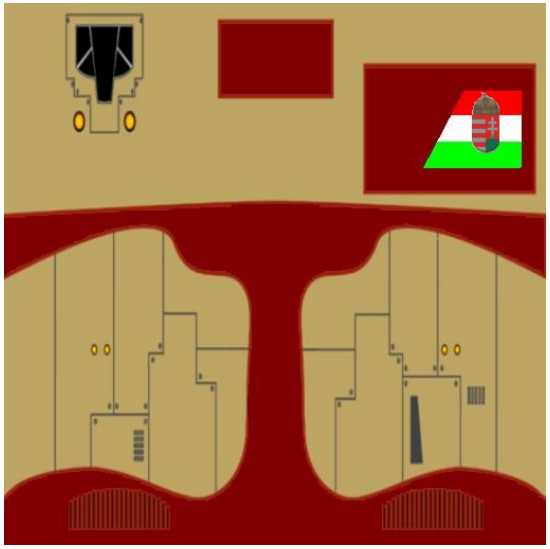


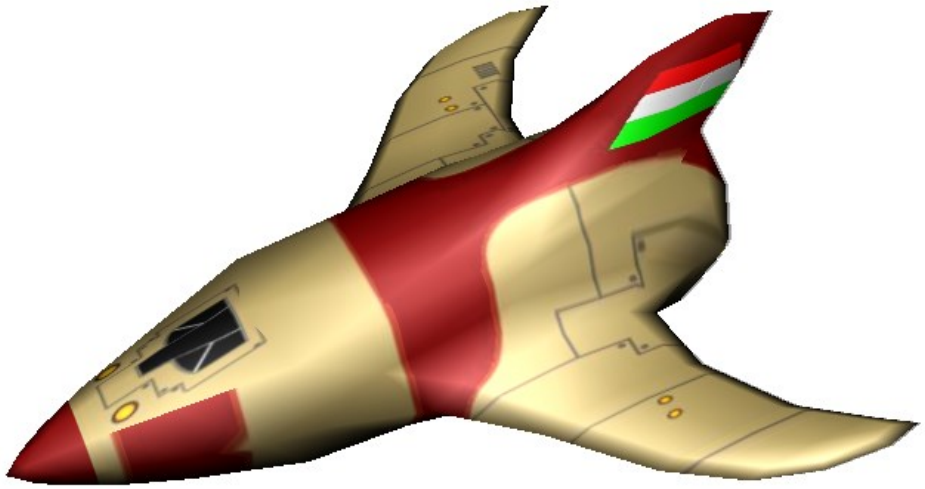
Űrhajó geometria: Catmull-Clark subdivision



Textúrához paraméterezés



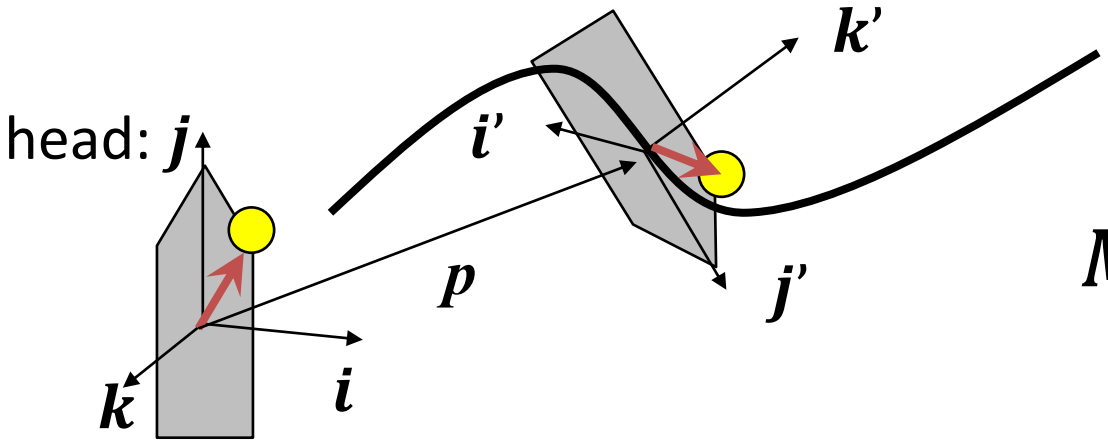




OBJ fájlformátum

```
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



$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{i}' & 0 \\ \boldsymbol{j}' & 0 \\ \boldsymbol{k}' & 0 \\ \boldsymbol{p} & 1 \end{bmatrix}$$

Orientáció, nem ortonormált:

$$\boldsymbol{j}' = \boldsymbol{v},$$
$$\boldsymbol{k}^* = \boldsymbol{k}'(1-\alpha) + \boldsymbol{a}\alpha,$$
$$\boldsymbol{i}' = \boldsymbol{j}' \times \boldsymbol{k}^*$$

Gram-Schmidt ortogonalizáció:

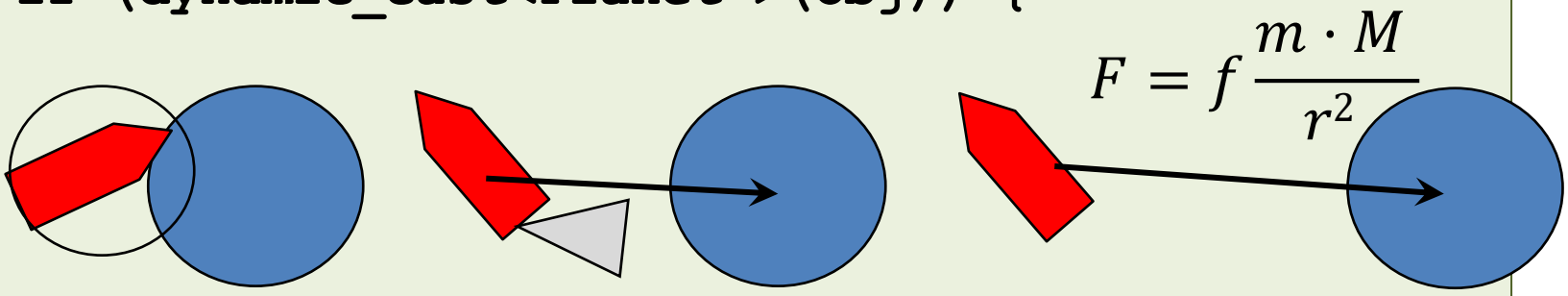
$$\boldsymbol{j}' = \boldsymbol{v}/|\boldsymbol{v}|,$$
$$\boldsymbol{i}' = \boldsymbol{j}' \times \boldsymbol{k}^*/|\boldsymbol{j}' \times \boldsymbol{k}^*|,$$
$$\boldsymbol{k}' = \boldsymbol{i}' \times \boldsymbol{j}'$$

Dinamikai szimuláció:

$$\boldsymbol{F} = m\boldsymbol{a}, \quad \boldsymbol{a} = \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}, \quad \boldsymbol{v} = \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} \Rightarrow \boxed{\boldsymbol{a} = \frac{\boldsymbol{F}}{m}, \quad \boldsymbol{v} \mathrel{+}= \boldsymbol{a}\mathrm{d}t, \quad \boldsymbol{p} \mathrel{+}= \boldsymbol{v}\mathrm{d}t}$$

Ship :: Control

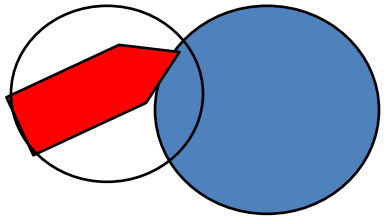
```
void Ship :: Control( float dt ) {
    force = vec3(0, 0, 0);
    for (GameObject * obj : objects) {
        if (dynamic_cast<Planet*>(obj)) {
```




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

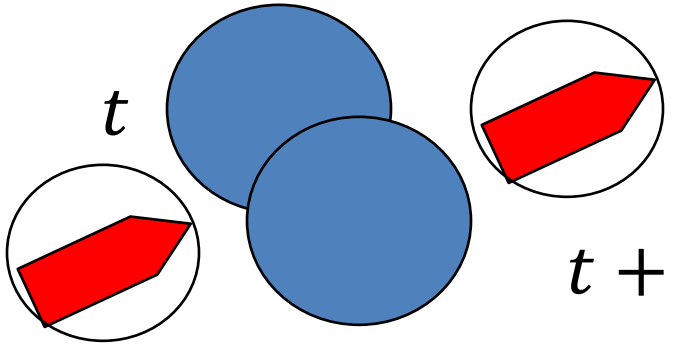

```
        }
    }
}
```

Ütközésdetektálás: lassú objektumok



adott t

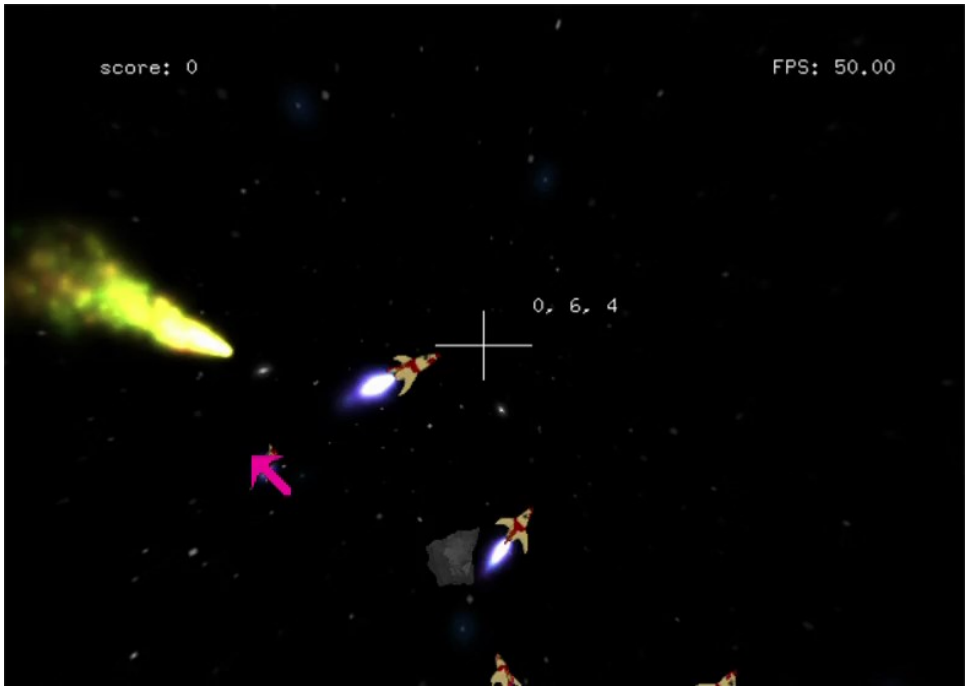
Probléma, ha az objektum gyors



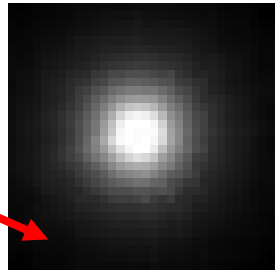

```
dist = length(obj1.pos - obj2.pos)
minDist = obj1.BoundingRadius() + obj2.BoundingRadius()
if (dist < minDist) Collision!
```

Foton torpedó

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



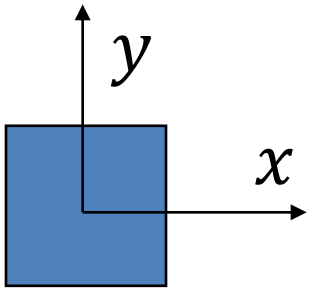
átlátszó

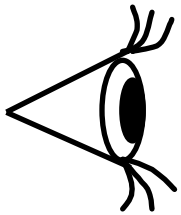


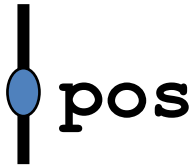
- Ütközésdetektálás = gyors mozgás

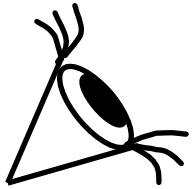
Billboard

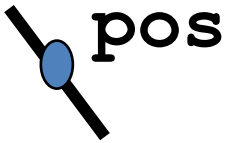
Egyetlen félig átlátszó textúra egy téglalapon

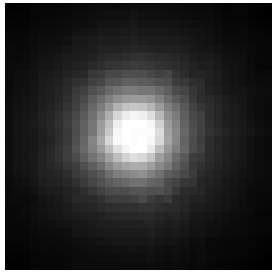


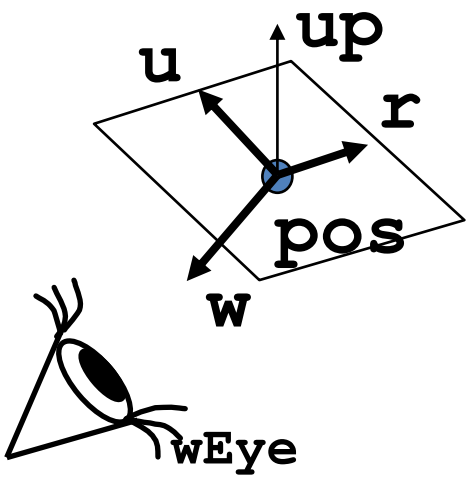










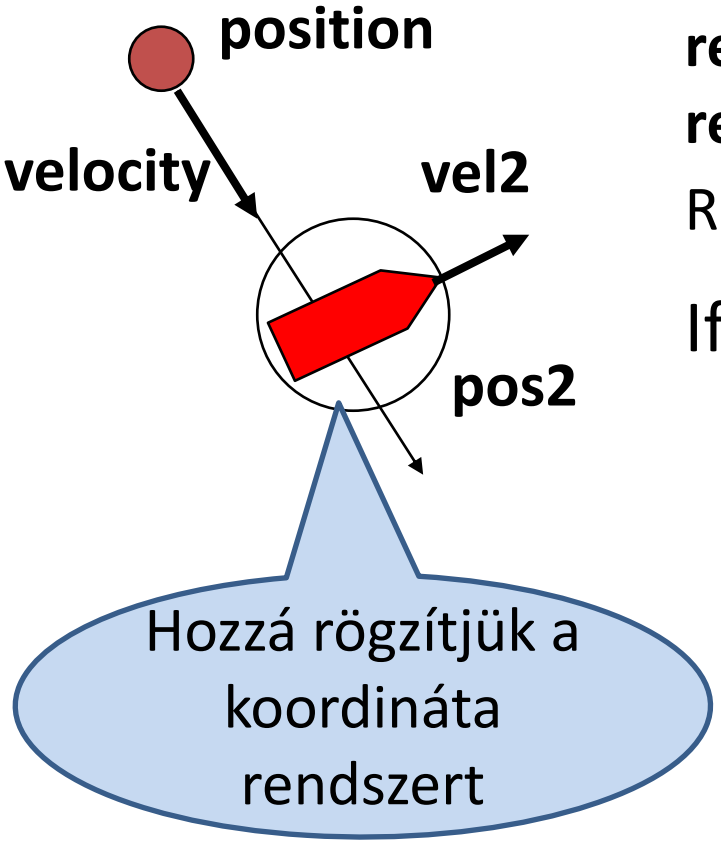



```
vec3 w = wEye - pos;
vec3 r = cross(w, up);
vec3 u = cross(r, w);
r = normalize(r) * size;
u = normalize(u) * size;
```

Billboard

```
void Bullet :: Draw(RenderState state) {
   vec3 up = vec3(0, 1, 0);
   vec3 w = state.wEye - pos;
   vec3 r = cross(w, up);
   vec3 u = cross(r, w);
   r = normalize(r) * size;
   u = normalize(u) * size;
   glEnable(GL_BLEND);          // átlátszóság
   glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
   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);
}
```

Gyors (folytonos) ütközés detektálás



rel_pos = **position** - **pos2**

rel_velocity = **velocity** - **vel2**

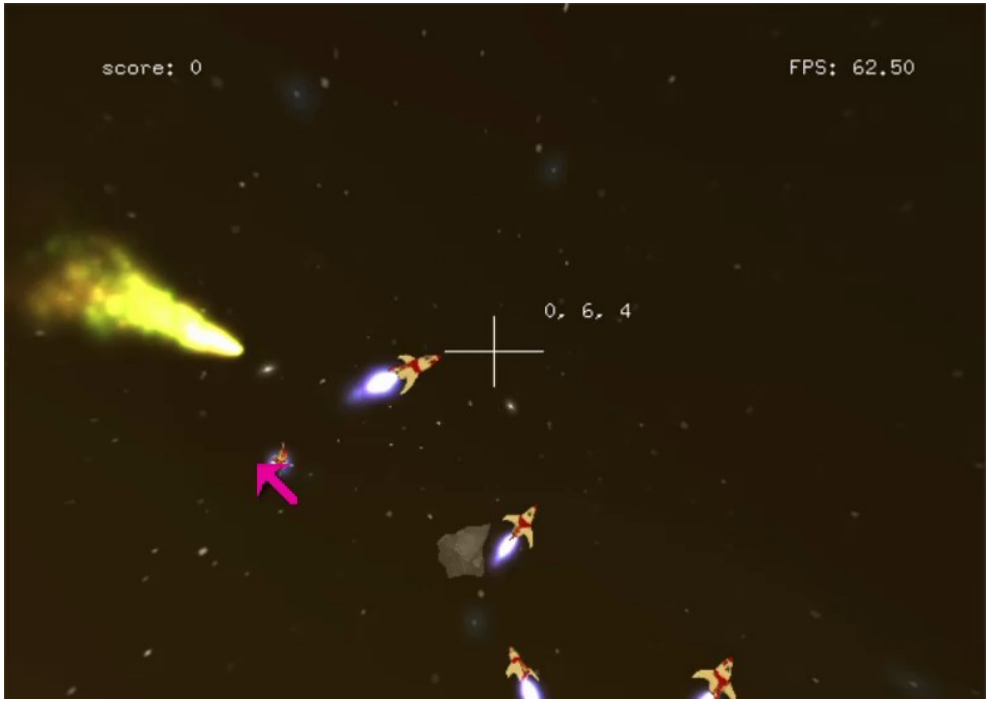
Ray: **rel_pos** + **rel_velocity**·t

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

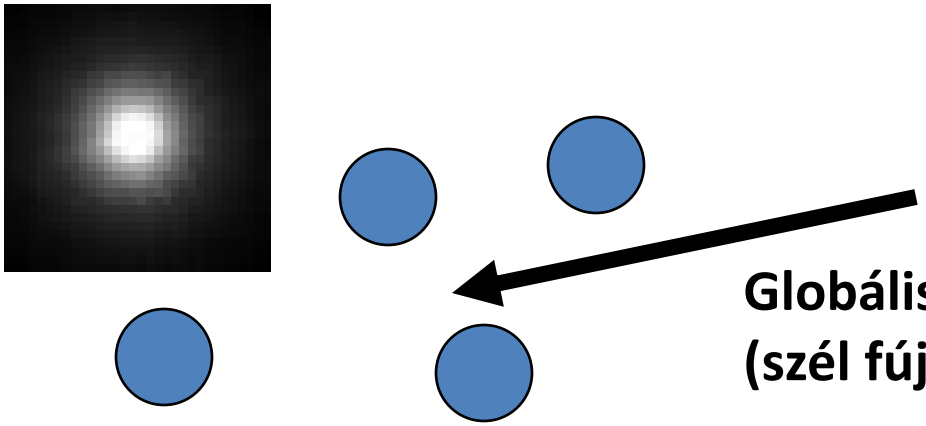
Robbanás

- Nagyon komplex geometria
- Hasonló kinézet minden irányból
- Plakátgyűjtemény

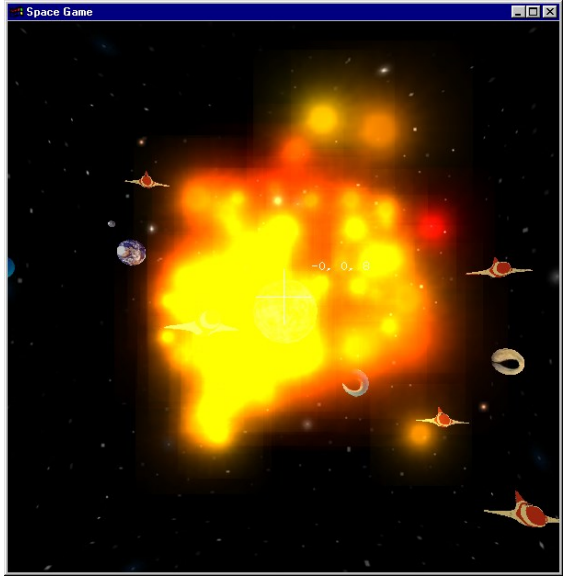
- Részecske rendszer



Részecske rendszerek



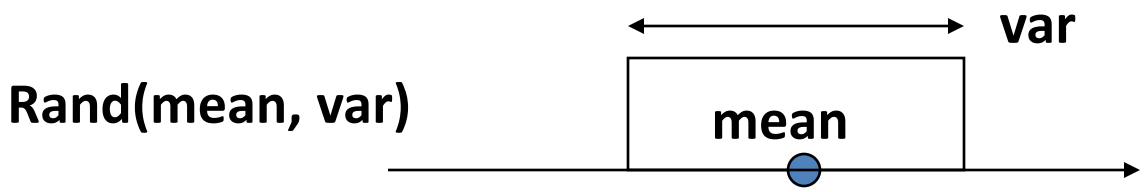
Globális erőtér
(szél fújja a füstöt)



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



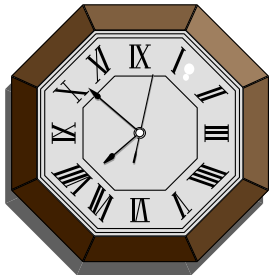

```
pos = center;                          // kezdetben fókuszált
lifetime = Rand(2, 1);

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;
```

Klaviatúra kezelés

KeyboardFunc
KeyboardUpFunc
SpecialKeysFunc
SpecialKeysUpFunc

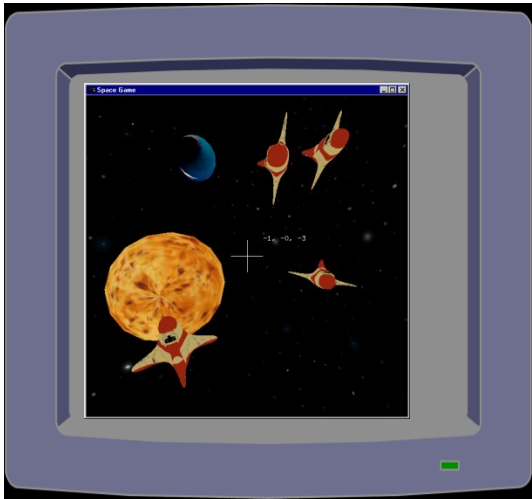
keys

```
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;
}
```

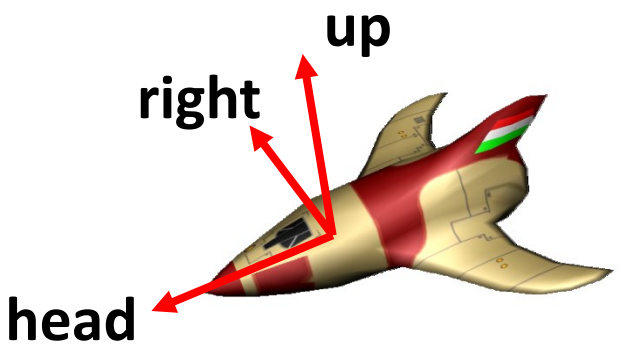


IdleFunc:
GameLoop

virtual
world



Player




```
class Player : public Avatar, public Ship {
  void ProcessInput() {
     if ( app.keys[' '] ) // 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 (app.keys[KEY_UP])    force -= up;
     if (app.keys[KEY_DOWN])  force += up;
     if (app.keys[KEY_LEFT])  force -= right;
     if (app.keys[KEY_RIGHT]) force += right;
  }
};
```