SolarSys

Florescu Bogdan-Ilie

Nedelcu Dragoș-Ioan

Vulpoi Gabriel

Roman Mateea

1.Conceptul proiectului:

Un univers prin care te poți plimba liber, descoperind pe parcurs noi sisteme solare, cu un număr aleator de planete, ce au un număr aleator de sateliți. Sateliții se învârt în jurul planetei, iar planetele în jurul soarelui. Ai trecut de un sistem solar pe care nu ai apucat sa îl admiri? Nicio problemă! Te poți întoarce la el, acesta pastrându-și poziția în spațiu. Mai mult, îl poți admira îndeaproape datorită funcției de zoom.

2.Transformări utilizate:

- Proiecția pe ecran este realizată cu glm::ortho.
- Pentru rotirea planetelor și a sateliților am utilizat glm::rotate.
- Acestea sunt poziționate în univers folosind glm::translate și redimensionate cu glm::scale.
- Camera este controlată utilizând glm::translate, iar pentru zoom modificăm proiecția pe ecran (valorile glm::ortho).

3.De ce este original proiectul ?

- **Universul este infinit:** Pentru sisteme solare am utilizat generarea procedurală.

- **Aspectul suprafețelor planetelor este generat in timp real:** Imaginile de pe planete sunt generate la runtime cu shadere

4.Contribuția Individuală:

- Florescu Bogdan-Ilie a realizat engine-ul;
- Nedelcu Dragos-Ioan a realizat generarea procedurală a sistemelor;
- Vulpoi Gabriel a realizat shadere și sistemul de entități;
- Roman Mateea a realizat render-ul.

Explicații Suplimentare:

În cele ce urmează va fi explicată mișcarea camerei:

1.În funcție de inputul de la tastatură se schimbă variabile ce țin de poziția camerei în spațiu și fov-ul acesteia:

```
if (Keys[SolarFuel::_Current]['D'])
{
    Position.x += GetTimeStep() * _Speed;
}

if (Keys[SolarFuel::_Current]['A'])
{
    Position.x -= GetTimeStep() * _Speed;
}

if (Keys[SolarFuel::_Current]['W'])
{
    Position.y += GetTimeStep() * _Speed;
}

if (Keys[SolarFuel::_Current]['S'])
{
    Position.y -= GetTimeStep() * _Speed;
}

if (Keys[SolarFuel::_Current]['Q'])
{
    Zoom += GetTimeStep() * _ZoomSpeed;
}

if (Keys[SolarFuel::_Current]['E'])
{
    Zoom -= GetTimeStep() * _ZoomSpeed;

    if (Zoom < _ZoomMin)
    {
        Zoom = _ZoomMin;
    }
}

ElapsedTime += GetTimeStep();
```

2. Aceste valori sunt transmise mai departe ca parametri pentru metoda de renderer ce se ocupă de update-ul de scenă:

```
_ActiveCamera.Position = Position;
_ActiveCamera.Angle = 0.0f;
_ActiveCamera.FieldOfView = Zoom;

_Renderer.StartScene(_ActiveCamera, _AspectRatio, ElapsedTime);
```

3.În metoda StartScene, se schimbă proiecția pe ecran, apelându-se metode ce alterează matricile de vizualizare și proiecție ortografică, în funcție de poziția camerei și fov-ul acesteia, schimbate anterior.

```
void SolarFuel::Graphics::Renderer::StartScene(const Camera& _ActiveCamera, const float _AspectRatio, const float _ElapsedTime)
{
    ProjectionView = _ActiveCamera.GetProjectionMatrix(_AspectRatio) * _ActiveCamera.GetViewMatrix();
    RenderObjects.clear();
    ElapsedTime = _ElapsedTime;
}
```

4.GetViewMatrix se ocupă de mișcarea camerei: această mișcare este realizată translatând obiectele în sens opus față de poziția nouă a camerei (i.e. față de direcția mișcării); GetProjectionMatrix realizează efectul de zoom, prin restrângerea limitelor câmpului vizual.

```
const glm::mat4 SolarFuel::Graphics::Camera::GetViewMatrix() const
{
    return glm::rotate(glm::mat4(1.0f), glm::radians(Angle), glm::vec3(0.0f, 0.0f, 1.0f)) * glm::translate(glm::mat4(1.0f), glm::vec3(-Position.x, -Position.y, 0.0f));
}

const glm::mat4 SolarFuel::Graphics::Camera::GetProjectionMatrix(const float _AspectRatio) const
{
    return glm::ortho(-FieldOfView / 2.0f * _AspectRatio, FieldOfView / 2.0f * _AspectRatio, -FieldOfView / 2.0f, FieldOfView / 2.0f);
}
```