Laboratorul 2

Grafică de bază

1 Generarea de geometrie

Acest tutorial explică în detaliu modațitățile prin care se poate genera geometrie procedurală în Unity. Sunt prezentate două abordări, una simplă și alta avansată în care memoria este manipulată manual. În acest laborator puteți folosi metoda simplă de a genera geometrie. Dacă aveți nevoie de așa ceva la proiecte, puteți folosi metoda avansată + *Burst*. În tutorial sunt generate și informații precum normale, tangente, coordonate de textură. Acestea nu fac parte din obiectivele acestui laborator, dar sunt foarte utile.

2 Shader Graph

Secțiunea 3.4 a acestui tutorial explică modul de definire al unui *Shader Graph* folosind URP. Secțiunea 2.1 a acestui tutorial arată modul de definire a unui nod custom din *Shader Graph* folosind *HLSL*.

3 Cerințe de laborator

3.1 Generarea unui pătrat (0.05p bonus)

Să se genreze în cod geometria unui pătrat format din două triunghiuri. Acesta va fi afișat pe ecran folosind un material care nu ia în calcul iluminarea scenei.

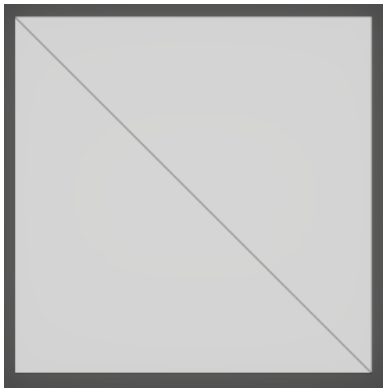

Figure 1: Pătrat format din două triunghiuri.

3.2 Subdivizarea unui pătrat (0.05p bonus)

Să se genereze în cod geometria unui pătrat format din $n \times n$ sub-pătrate `FĂRĂ A AVEA VERTECȘI DUPLICAȚI`. Sub-pătratele vor fi formate din câte două triunghiuri.

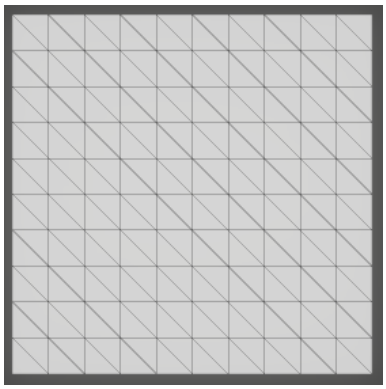

Figure 2: Pătrat format din 10×10 sub-pătrate.

3.3 Generarea unui cub (0.1p bonus)

Să se genereze 6 fețe cu rezoluția $n \times n$ care să formeze un cub. Fețele trebuie să facă parte din `ACEEAȘI GEOMETRIE` (nu folosițe $Mesh$-uri separate).

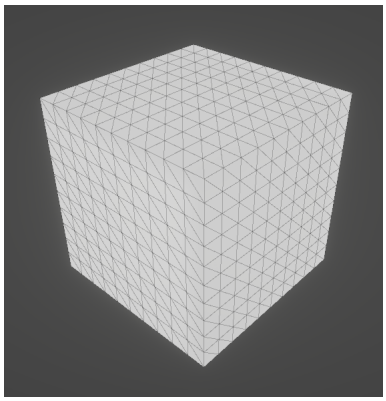

Figure 3: Cub format din 6 fețe cu rezoluția 10×10.

3.4 Sferă cub (0.05p bonus)

Să se aplice metoda `Vector3.Normalize` asupra fiecărui vertex al cubului pentru a-l transforma într-o sferă cu raza 1. Acest tip de sferă se numește `Sferă Cub` (*Cube Sphere* în engleză).

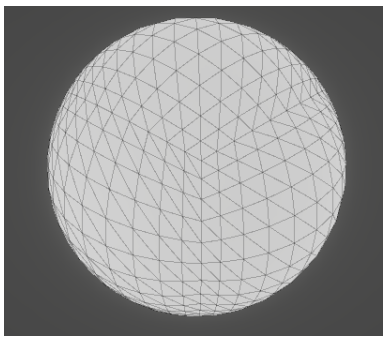

Figure 4: Cub transformat în sferă.

3.5 Distorsionarea vertecșilor (0.05p bonus)

Să se modifice poziția vertecșilor sferei folosind un zgomot de tip gradient (pentru orice punct din spațiu, se poate calcula o valoare cuprinsă între -1 și 1 care pare aleatoare, dar este foarte apropiată de valorile obținute pentru punctele din apropiere). Puteți folosi această implementare 3D pentru Perlin Noise (*Unity* implementează doar variantele 1D și 2D pentru *Perlin Noise*, `Mathf.PerlinNoise`). Dacă folosiți această implementare, atunci puteți modifica poziția unui vertex în felul următor:

```
vertex = vertex * (1.0f + Perlin.Noise(vertex.x, vertex.y, vertex.z) * _displacement);
```

Unde `_displacement` reprezintă o valoare între 0 și 1 care reprezintă factorul de distorsionare.

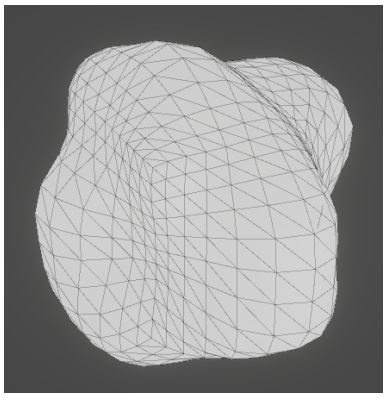

Figure 5: Sferă cu vertecși distorsionați.

3.6 Shader custom (0.1p bonus)

Să se definească un shader prin care să se coloreze obiectul generat anterior în modul următor:

$$BaseColor = \begin{cases} \frac{length(pos_{obj})-1}{Displacement} color_A, & \text{for } length(pos_{obj}) \geq 1 \\ \frac{1-length(pos_{obj})}{Displacement} color_B, & \text{for } length(pos_{obj}) < 1 \end{cases}$$

Unde

- $BaseColor$ este culoarea finală.
- pos_{obj} este poziția în *Object Space* a vertexului.
- $Displacement$ este factorul de distorsionare folosit la punctul anterior.
- $color_A$ este culoarea cu care vrem să fie colorată sfera în zonele superioare.
- $color_B$ este culoarea cu care vrem să fie colorată sfera în zonele inferioare.

Se va porni de la un *Unlit Shader Graph* și se va calcula culoarea finală folosind formula definită mai sus. Se pot folosi fie mai multe noduri, fie un singur nod care să conțină o funcție custom scrisă în $HLSL$.

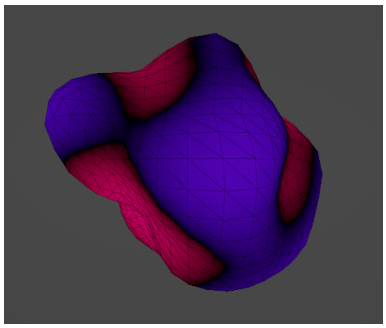

Figure 6: Zonele superioare ale sferei sunt colorate folosind o culoare, iar cele inferioare sunt colorate folosind altă culoare. Intensitatea culorii depinde de distorsionare.