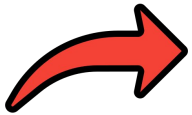
Eksplozje z użyciem shadera geometrycznego

Paweł Wójcik, Damian Wojtyczko

Wstęp do geometry shadera

Shader geometryczny możemy wyobrażać sobie jako pośrednika pomiędzy shaderem wierzchołków, a shaderem fragmentów. Jego głównym celem jest manipulowanie prymitywami (takimi jak punkty lub trójkąty) przed przesłaniem ich do shadera fragmentów. Poniższy przykład przesyła tylko punkt dalej i nic z nim nie robi. Ma to na celu zaprezentowanie składni i schematu budowy takiego shadera.



```
#version 330 core
layout(location = 0) in vec3 pos;

void main() {
    gl_Position = vec4(pos, 1.0);
}
```

```
#version 330 core
layout(points) in;
layout(points, max_vertices = 1) out;

void main() {
    gl_Position = gl_in[0].gl_Position;
    EmitVertex();
    EndPrimitive();
}
```

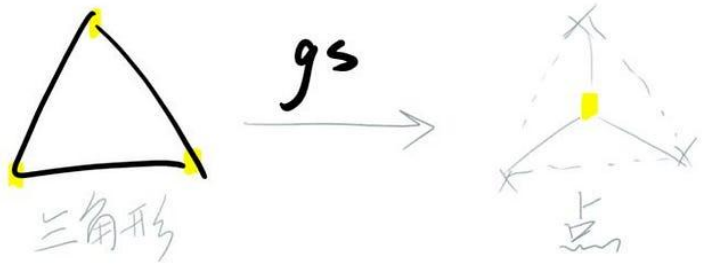
```
#version 330 core
out vec4 fragColor;

void main() {
    fragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Wykorzystanie geometry shadera

Wykorzystanie geometry shadera sprowadza się do dwóch głównych celów:

- Generowanie nowych prymitywów,
- Redukcja istniejących prymitywów.

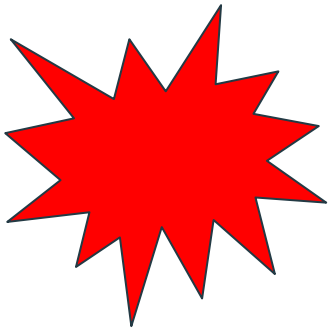
W kontekście symulacji eksplozji, chcemy przekształcać trójkąty obecne w plikach 3D (np. .obj) na pojedyncze punkty w przestrzeni. Zmniejsza to liczbę wierzchołków i ułatwia manipulowanie nimi, co jest kluczowe przy symulacji eksplozji.



Symulacja eksplozji

W celu dokładnej symulacji eksplozji potrzebujemy wykonać trzy kluczowe kroki:

- **Zamiana trójkątów na punkty:**
  - Na etapie geometry shadera, zamieniamy trójkąty na wierzchołki w celu łatwiejszej symulacji.
- **Obliczenie normalnych**:
  - Kierunek ruchu od środka eksplozji.
- **Symulacja ruchu**
  - Obliczanie przemieszczenia wierzchołków inspirowane podstawową kinetyką.



Symulacja eksplozji - trójkąty

Aby uzyskać zadowalający efekt, musimy przekształcić trójkąty otrzymane od vertex shadera na wierzchołki. W geometry shaderze zamieniamy każdy trójkąt na jego trzy składowe wierzchołki i emitujemy ich środek ciężkości jako osobny prymityw. Dzięki temu mamy nad nimi pełną swobodę podczas modelowania eksplozji. Tak przygotowane obiekty modyfikujemy dalszych kroków.

```
#version 330 core

layout(triangles) in;
layout(points, max_vertices = 3) out;

in  vec3 worldPos[];

vec3 GetCenter()
{
    return (worldPos[0] + worldPos[1] + worldPos[2]) / 3.0;
}
```

Symulacja eksplozji - normalne

Załóżmy, że gl_in zawiera trzy wierzchołki trójkąta w przestrzeni 3D (przekazane z vertex shadera). Aby obliczyć jego normalną, wyznaczamy wektory a i b reprezentujące 2 boki trójkąta zawieszone w tym samym wierzchołku.

Normalna to wtedy znormalizowany iloczyn wektorowy a i b.

```
layout(triangles) in;
layout(points, max_vertices = 3) out;

in  vec3 worldPos[];

vec3 GetNormal()
{
    vec3 a = worldPos[1] - worldPos[0];
    vec3 b = worldPos[2] - worldPos[0];
    return normalize(cross(a, b));
}
```

Symulacja eksplozji - ruch

W celu zamodelowania ruchu poszczególnych wierzchołków, inspirujemy się podstawami kinematyki. Pozycja każdego wierzchołka zmienia się zgodnie ze wzorem na przemieszczenie.

Implementacja wykorzystuje normalne jako kierunek ruchu (od środka eksplozji) i na tej podstawie oblicza przemieszczenie zależne od czasu.

$$s = v_0 t + \frac{1}{2} a t^2$$

```
vec4 explode(vec4 position, vec3 normal, float time)
{
    float v = 2.0; // prędkość
    float a = 0.5; // przyspieszenie
    float t = max(time, 0.0);

    vec3 direction = normal * (v * t + 0.5 * a * t * t);
    return position + vec4(direction, 0.0);
}
```

Prezentacja efektu