Компьютерная графика
Геометрические шейдеры

Лекция 11

Демяненко Я.М. ЮФУ 2025

Впервые появились

- Геометрические шейдеры были введены компанией Microsoft в DirectX10, а затем были добавлены в ядро OpenGL версии 3.2.

Выполняются шейдеры для ...

- Вершинные — для вершин
- Тесселяционные — для патчей
- **Геометрические — для примитивов**
- Фрагментные — для пикселей

Особенности геометрических шейдеров

Связи между вершинами доступны разработчику

Геометрический шейдер **может изменять примитивы**

Изменение топологии входящих примитивов

Геометрический шейдер может:

- **принимать** примитивы **любого** типа,
- но **выводить** может только:
  - **списки точек**,
  - **стрип линий,**
  - **стрип треугольников**

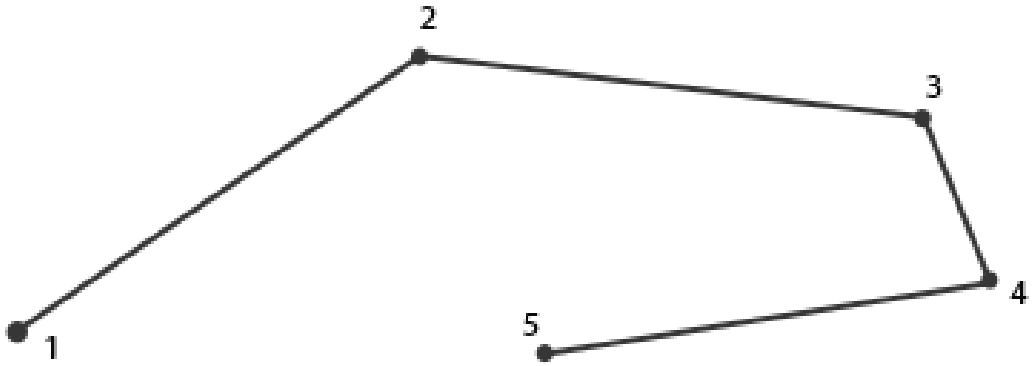
Growing Geometry

Геометрический шейдер

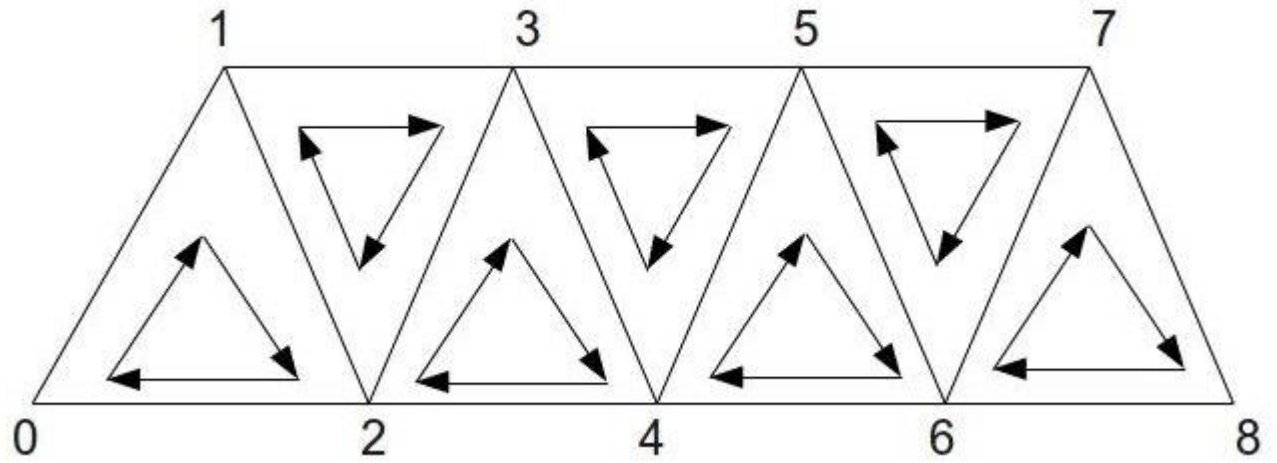
- принимает один примитив
- и может
  - либо удалить его полностью,
  - либо отправить на выход один или несколько примитивов

Это значит, что он **может выпускать и меньше и больше вершин, чем получает**. Эта способность известна как **growing geometry**.

Line Strip



Стрип треугольников



Важное свойство касательно порядка внутри треугольника — порядок обратен у нечётных треугольников.

Это значит, что порядок таков: [0,1,2], [1,3,2], [2,3,4], [3,5,4] и т.д.

Пример геометрического шейдера

```
#version 330 core
layout (points) in;
layout (line_strip, max_vertices = 2) out;

void main() {
    gl_Position = gl_in[0].gl_Position + vec4(-0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    gl_Position = gl_in[0].gl_Position + vec4( 0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    EndPrimitive();
}
```

layout(input_primitive) in;

points:	При выводе GL_POINTS (1)
lines:	При выводе GL_LINES или GL_LINE_STRIP (2)
lines_adjacency:	При выводе GL_LINES_ADJACENCY или GL_LINE_STRIP_ADJACENCY (4)
triangles:	При выводе GL_TRIANGLES, GL_TRIANGLE_STRIP или GL_TRIANGLE_FAN (3)
triangles_adjacency :	При выводе GL_TRIANGLES_ADJACENCY или GL_TRIANGLE_STRIP_ADJACENCY (6)

Пример геометрического шейдера

```
#version 330 core
layout (points) in;
layout (line_strip, max_vertices = 2) out;

void main() {
    gl_Position = gl_in[0].gl_Position + vec4(-0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    gl_Position = gl_in[0].gl_Position + vec4( 0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    EndPrimitive();
}
```

layout(output_primitive, max_vertices = vert_count) out;

- points
- line_strip
- triangle_strip

Пример геометрического шейдера

```
#version 330 core
layout (points) in;
layout (line_strip, max_vertices = 2) out;

void main() {
    gl_Position = gl_in[0].gl_Position + vec4(-0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    gl_Position = gl_in[0].gl_Position + vec4( 0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    EndPrimitive();
}
```

Встроенная переменная gl_in

```
//Приблизительное представление

in gl_Vertex {
    vec4  gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[];
} gl_in[];
```

Пример геометрического шейдера

```
#version 330 core
layout (points) in;
layout (line_strip, max_vertices = 2) out;

void main() {
    gl_Position = gl_in[0].gl_Position + vec4(-0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    gl_Position = gl_in[0].gl_Position + vec4( 0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    EndPrimitive();
}
```

Каждый вызов **EmitVertex()** добавляет текущее значение в переменной **gl_Position** к текущему экземпляру примитива.

Когда же мы вызываем **EndPrimitive()**, все порожденные вершины окончательно связываются в указанный выходной тип примитива.

Пример геометрического шейдера

```
#version 330 core
layout (points) in;
layout (line_strip, max_vertices = 2) out;

void main() {
    gl_Position = gl_in[0].gl_Position + vec4(-0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    gl_Position = gl_in[0].gl_Position + vec4( 0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    EndPrimitive();
}
```

Повторяя вызовы **EndPrimitive()** после одного или более вызовов **EmitVertex()** можно продолжать создавать новые экземпляры примитивов.

Конкретно в примере генерируется по две вершины, смещенные на небольшое расстояние от положения входной вершины, а затем выполняется вызов **EndPrimitive()**, формирующий из этих двух сгенерированных вершин один **line strip**, содержащий две вершины.

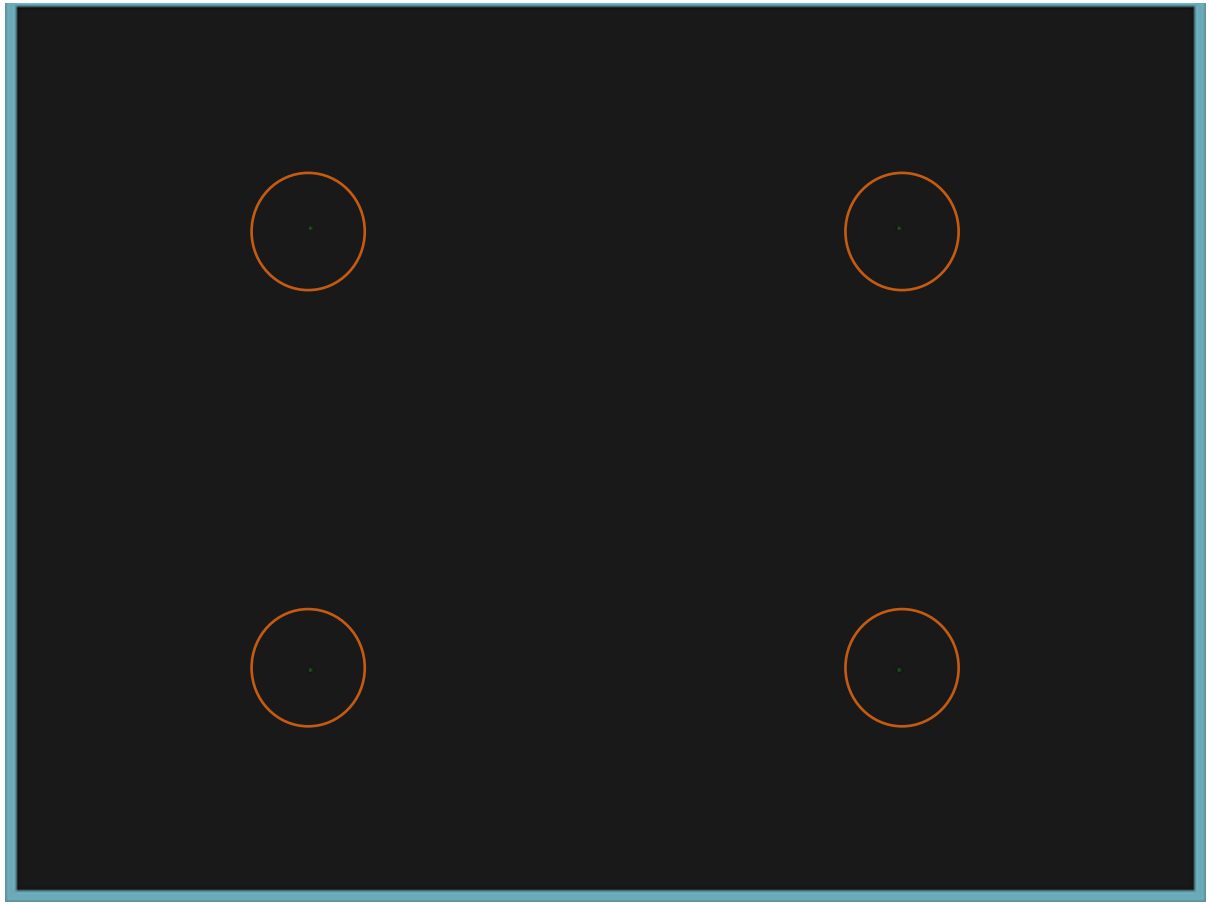
Получили такие результаты, выполнив всего лишь один вызов отрисовки:

glDrawArrays(GL_POINTS, 0, 4);

Компиляция и линковка

```
geometryShader = glCreateShader(GL_GEOMETRY_SHADER);
glShaderSource(geometryShader, 1, &gShaderCode, NULL);
glCompileShader(geometryShader);
...
glAttachShader(program, geometryShader);
glLinkProgram(program);
```

Использование геометрического шейдера



Входные данные

```
float points[] = {
        -0.5f,  0.5f, // верхняя-левая
         0.5f,  0.5f, // верхняя-правая
         0.5f, -0.5f, // нижняя-правая
        -0.5f, -0.5f  // нижняя-левая
};
```

Вершинный шейдер

```
#version 330 core
layout (location = 0) in vec2 aPos;

void main() {
    gl_Position = vec4(aPos.x, aPos.y, 0.0, 1.0);
}
```

Геометрический шейдер, который просто берет данные входного примитива и отправляет на выход без изменений

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

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

Фрагментный шейдер

```
#version 330 core
out vec4 FragColor;

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

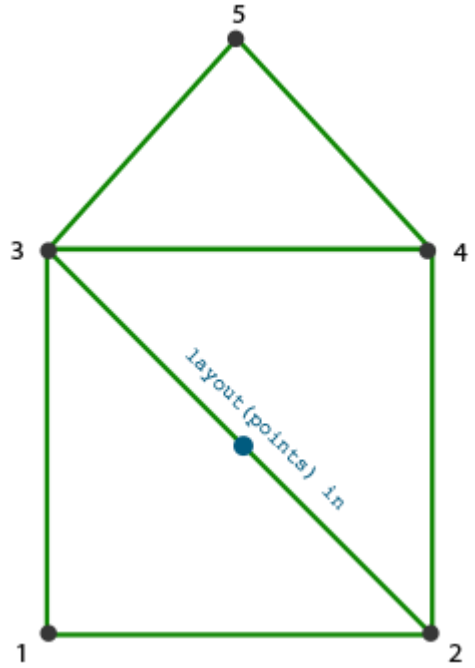
В коде программы

Создаем VAO и VBO для вершинных данных и осуществляем рендер вызовом glDrawArrays():

```
shader.use();
glBindVertexArray(VAO);
glDrawArrays(GL_POINTS, 0, 4);
```

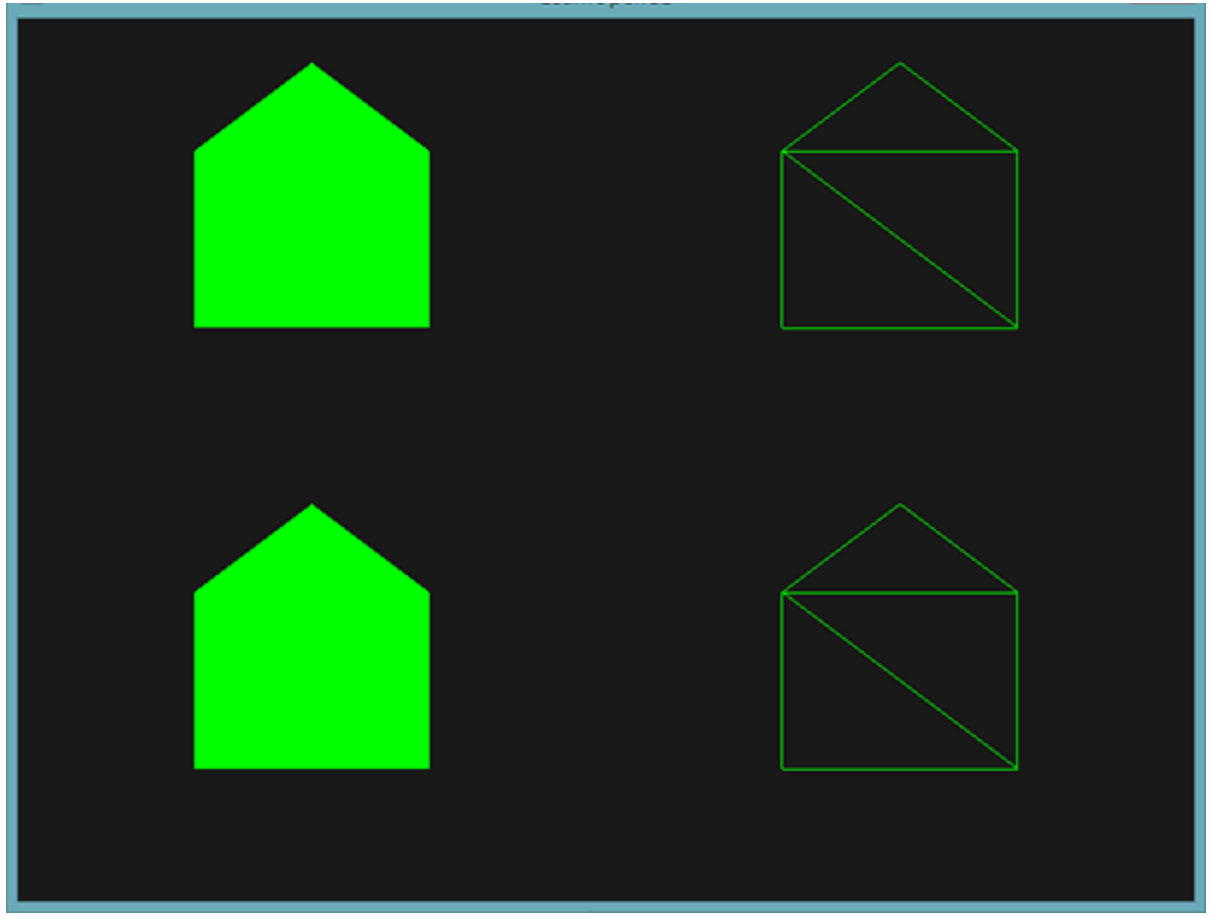
Строим домики

Понадобится сменить тип выходного примитива **на triangle_strip** и нарисовать три треугольника: два для создания квадратной основы и один для крыши



Результирующий геометрический шейдер

```
#version 330 core
layout (points) in;
layout (triangle_strip, max_vertices = 5) out;

void build_house(vec4 position) {
    gl_Position = position + vec4(-0.2, -0.2, 0.0, 0.0);    // 1:bottom-left
    EmitVertex();
    gl_Position = position + vec4( 0.2, -0.2, 0.0, 0.0);    // 2:bottom-right
    EmitVertex();
    gl_Position = position + vec4(-0.2,  0.2, 0.0, 0.0);    // 3:top-left
    EmitVertex();
    gl_Position = position + vec4( 0.2,  0.2, 0.0, 0.0);    // 4:top-right
    EmitVertex();
    gl_Position = position + vec4( 0.0,  0.4, 0.0, 0.0);    // 5:top
    EmitVertex();
    EndPrimitive();
}
void main() {
    build_house(gl_in[0].gl_Position);
}
```



Обновленные данные вершин (с цветом)

```
float points[] = {
    -0.5f,  0.5f, 1.0f, 0.0f, 0.0f, // верхняя-левая
     0.5f,  0.5f, 0.0f, 1.0f, 0.0f, // верхняя-правая
     0.5f, -0.5f, 0.0f, 0.0f, 1.0f, // нижняя-правая
    -0.5f, -0.5f, 1.0f, 1.0f, 0.0f  // нижняя-левая
};
```

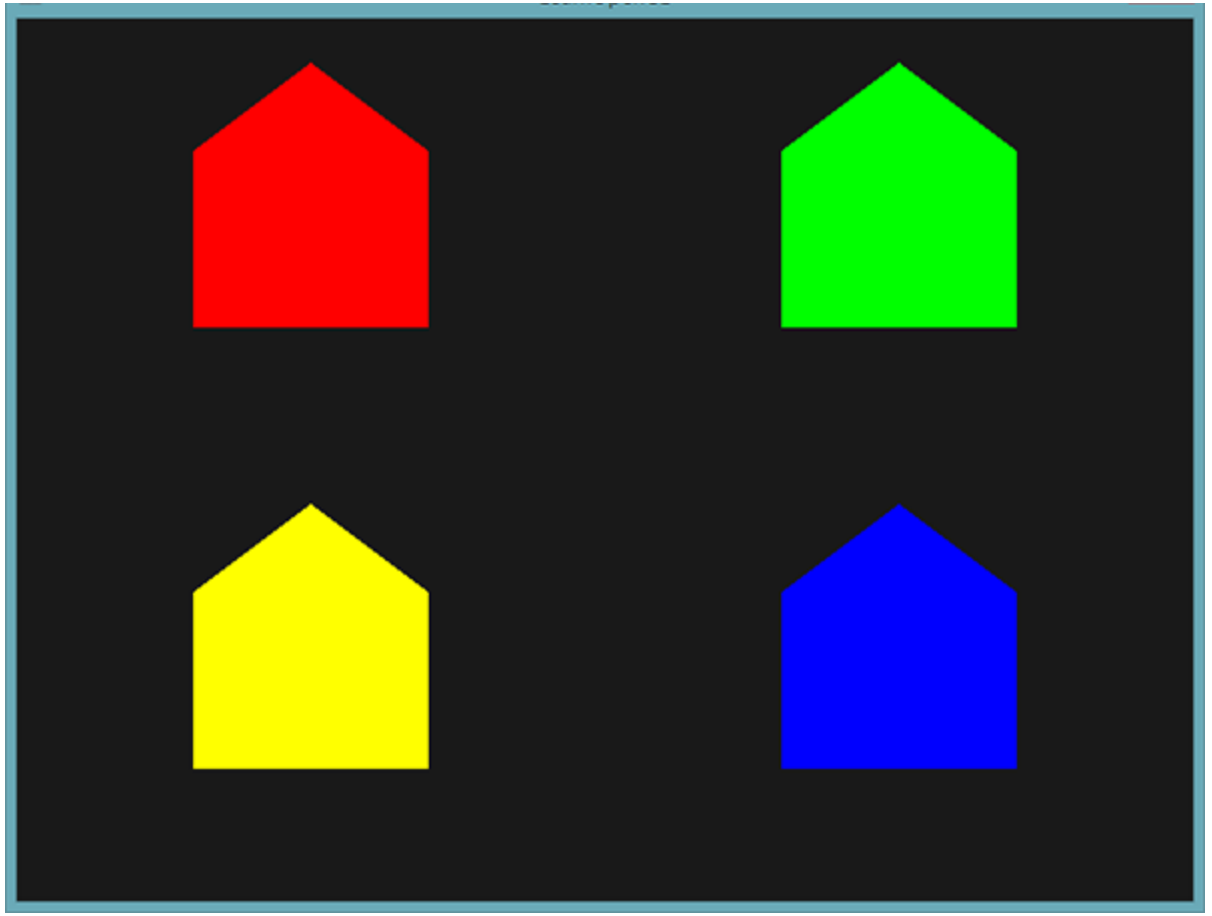
Вершинный шейдер для передачи атрибута цвета в геометрический шейдер с использованием интерфейсного блока

```
#version 330 core
layout (location = 0) in vec2 aPos;
layout (location = 1) in vec3 aColor;

out VS_OUT {
    vec3 color;
} vs_out;

void main() {
    gl_Position = vec4(aPos.x, aPos.y, 0.0, 1.0);
    vs_out.color = aColor;
}
```

Интерфейсный блок того же типа (но с другим именем) в геометрическом шейдере

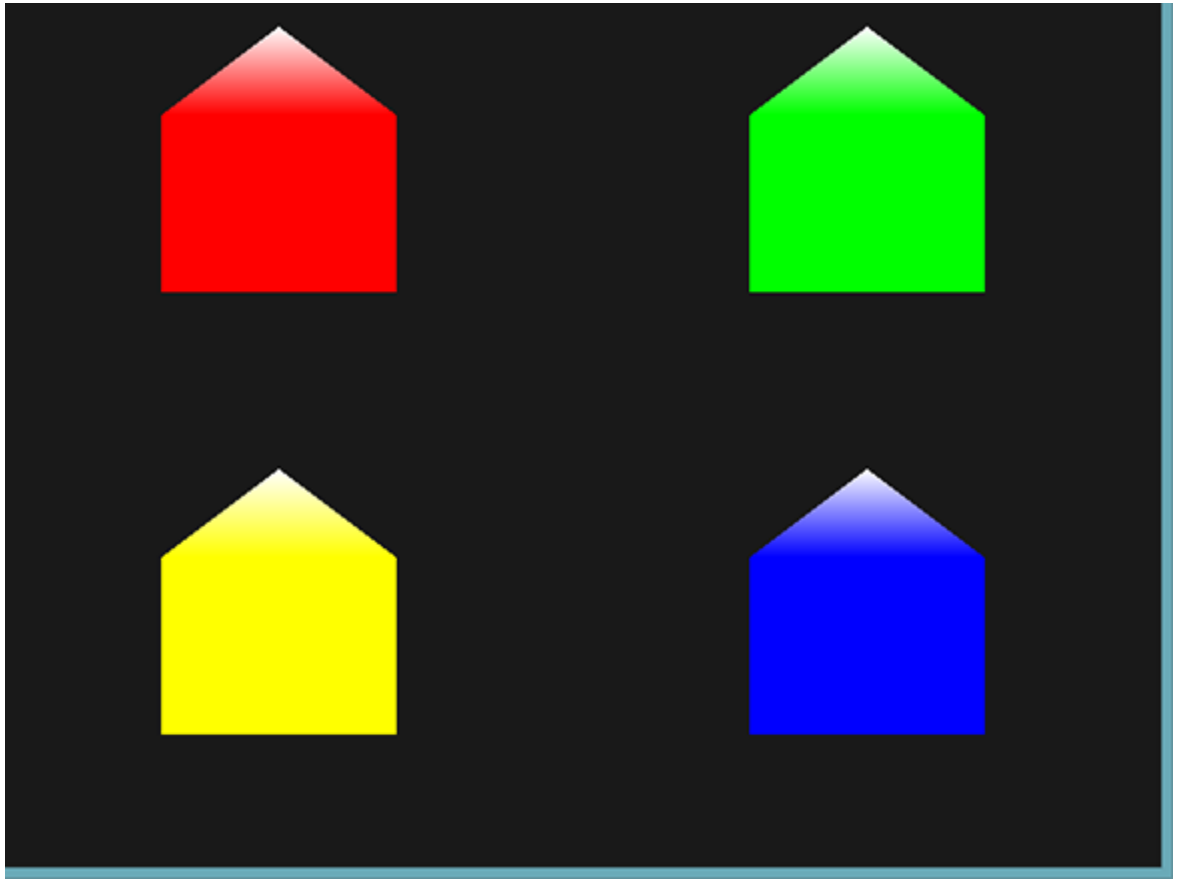
```
in VS_OUT {
    vec3 color;
} gs_in[];
```

```
out vec3 fColor;

…

fColor = gs_in[0].color; // используется gs_in[0] поскольку на входе у нас единственная вершина
gl_Position = position + vec4(-0.2, -0.2, 0.0, 0.0);    // 1:нижняя-левая
EmitVertex();
gl_Position = position + vec4( 0.2, -0.2, 0.0, 0.0);    // 2:нижняя-правая
EmitVertex();
gl_Position = position + vec4(-0.2,  0.2, 0.0, 0.0);    // 3:верхняя-левая
EmitVertex();
gl_Position = position + vec4( 0.2,  0.2, 0.0, 0.0);    // 4:верхняя-правая
EmitVertex();
gl_Position = position + vec4( 0.0,  0.4, 0.0, 0.0);    // 5:крыша
EmitVertex();
EndPrimitive();
```



Присыпим крыши домиков снегом

```
fColor = gs_in[0].color;
gl_Position = position + vec4(-0.2, -0.2, 0.0, 0.0);    // 1:нижняя-левая
EmitVertex();
gl_Position = position + vec4( 0.2, -0.2, 0.0, 0.0);    // 2:нижняя-правая
EmitVertex();
gl_Position = position + vec4(-0.2,  0.2, 0.0, 0.0);    // 3:верхняя-левая
EmitVertex();
gl_Position = position + vec4( 0.2,  0.2, 0.0, 0.0);    // 4:верхняя-левая
EmitVertex();
gl_Position = position + vec4( 0.0,  0.4, 0.0, 0.0);    // 5:крыша
fColor = vec3(1.0, 1.0, 1.0);
EmitVertex();
EndPrimitive();
```



Взрываем объекты

Перемещение каждого треугольника вдоль направления нормали с течением времени.

В результате этот эффект даёт подобие взрыва объекта, разделяя его на отдельные треугольники, движущиеся по направлению своего вектора нормали.

Использование геометрического шейдера позволяет эффекту работать на любом объекте, вне зависимости от его сложности

Вычисление вектора нормали по трём вершинам входного треугольника

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

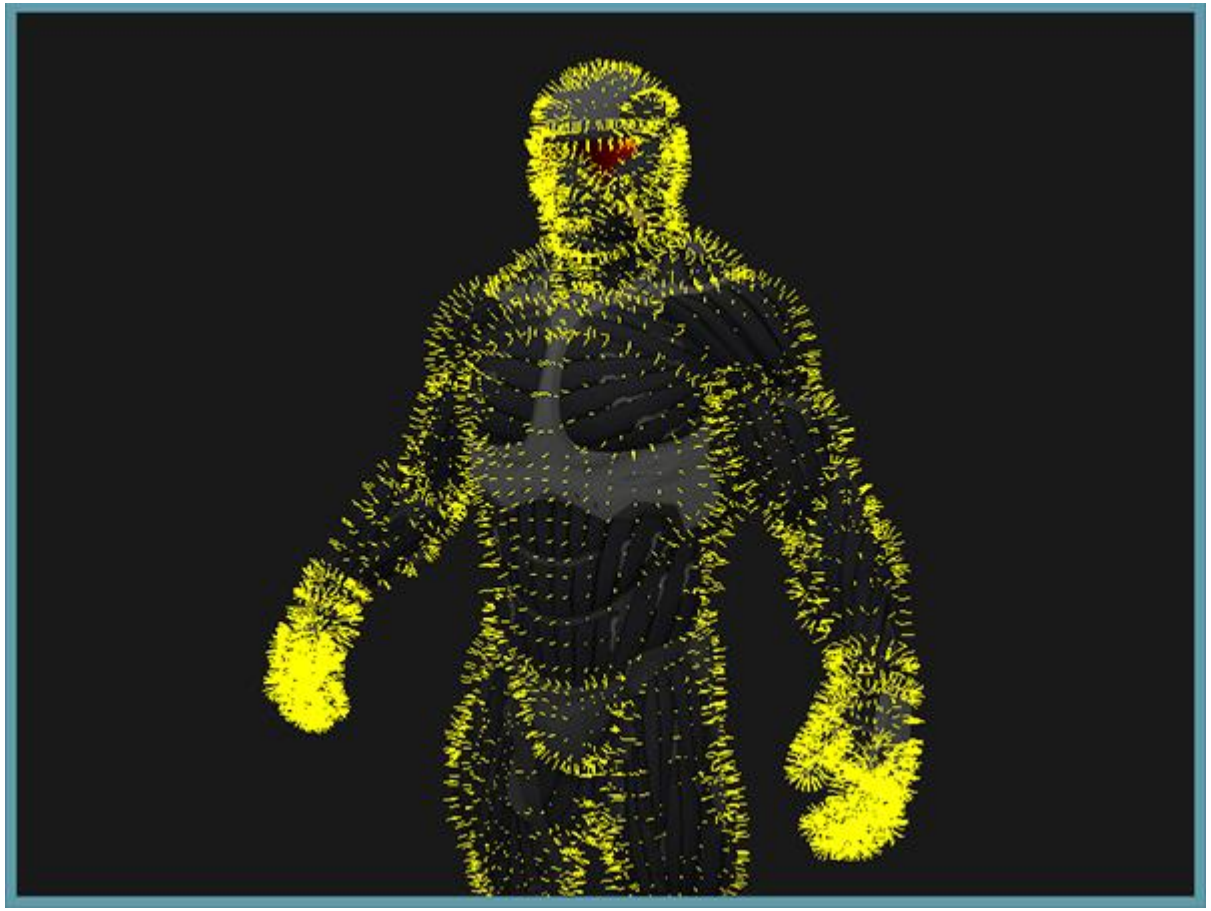
Функция принимает вектора нормали и положения вершины, а возвращает новое положение вершины, смещённое вдоль нормали

```
vec4 explode(vec4 position, vec3 normal) {
    float magnitude = 2.0;
    vec3 direction = normal * ((sin(time) + 1.0) / 2.0) * magnitude;
    return position + vec4(direction, 0.0);
}
```

main() геометрического шейдера

```
void main() {
    vec3 normal = GetNormal();

    gl_Position = explode(gl_in[0].gl_Position, normal);
    TexCoords = gs_in[0].texCoords;
    EmitVertex();
    gl_Position = explode(gl_in[1].gl_Position, normal);
    TexCoords = gs_in[1].texCoords;
    EmitVertex();
    gl_Position = explode(gl_in[2].gl_Position, normal);
    TexCoords = gs_in[2].texCoords;
    EmitVertex();
    EndPrimitive();
}
```

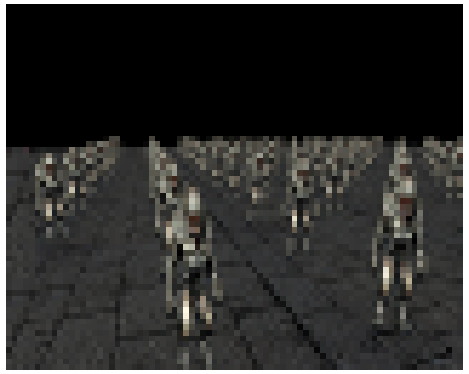
Визуализация нормалей

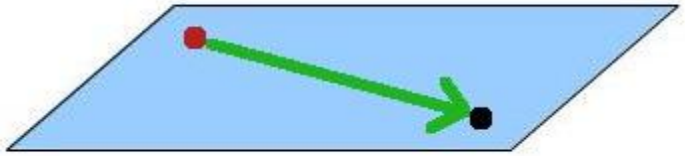


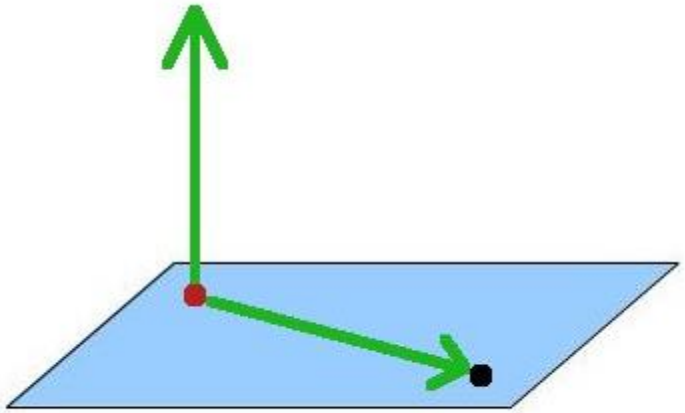
Реализация метода billboarding

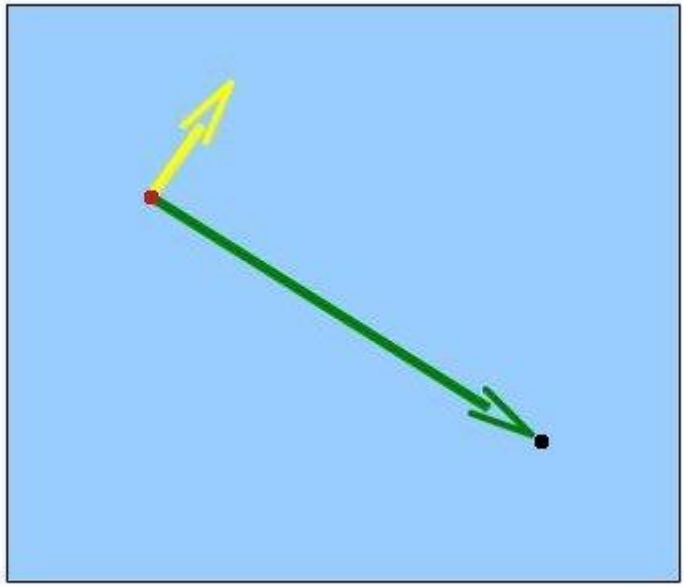
Billboarding - прямоугольник, который всегда направлен в камеру.

При движении камеры по сцене billboard вращается за ней так, что вектор из billboard до камеры всегда перпендикулярен поверхности billboard.









желтый вектор — результат векторного произведения

Процедурная растительность