Тесселяция

Компьютерная графика

План

- Что это такое?
- Мотивация
- Тесселяционный конвейер
- Практическое использование

Что это такое?

Тесселяция — это разбиение некоторого примитива на несколько меньших.

Мотивация

- Качество визуализации
- Экономия памяти
- Динамический уровень детализации
- Выполнение расчетов на уровне вершин

Base model

Bump mapping

Displacement mapping

Тесселяционный конвейер в OpenGL 4.0

Тесселяция в OpenGL 4.0

- Новый тип примитивов
  - Патчи (GL_PATCHES)

- Две программируемые стадии
  - Tessellation Control Shader (GL_TESS_CONTROL_SHADER)
  - Tessellation Evaluation Shader (GL_TESS_EVALUATION_SHADER)

- Одна фиксированная стадия
  - Tessellation Primitive Generator

Tessellation Control Шейдер. Что делает?

- Выполняется до тесселяции и задаёт ее параметры
- Выполняется для каждой контрольной вершины выходного патча, но при этом имеет полный доступ ко всем вершинам этого патча (как входным, так и выходным)

Задача Tessellation Control шейдера

- задать параметры для тесселятора и tessellation evaluation shader’a — атрибуты для патча
  - Тесселяционные факторы для разбиения
  - Пользовательские атрибуты

Патч

- Примитив с числом вершин от 1 до 32
- Включает в себя как вершинные атрибуты, так и атрибуты задаваемые для всего патча

Патч. Число вершин

- Число вершин задается
  - **или** в **tesselation control shader**
  - **или**, если этот шейдер отсутствует, в коде самого приложения, при помощи команды **glPatchParameteri**

В коде приложения

- glPatchParameteri(GL_PATCH_VERTICES, vertexCount)

- При использовании тесселяции допустимым типом примитива для glDrawArrays является только GL_PATCHES

no tessellation:
glDrawArrays(GL_TRIANGLES, firstVertex, vertexCount);

with tessellation:
glPatchParameteri(GL_PATCH_VERTICES, 3);
glDrawArrays(GL_PATCHES, firstVertex, vertexCount);

glPatchParameterfv(GL_PATCH_DEFAULT_OUTER_LEVEL, outerLevels)
glPatchParameterfv(GL_PATCH_DEFAULT_INNER_LEVEL, innerLevels)

Число вершин в примитиве

При помощи директивы layout в нем явно задается число вершин в примитиве.

layout(vertices = 3) out;

layout

- layout(домен, разбиение, топология)
  - triangles, quads, isolines
  - equal_spacing, fractional_even_spacing, fractional_odd_spacing
  - cc, ccw, point_mode

- layout(triangles, equal_spacing, ccw)

Особенности данных

Все входные параметры, пришедшие от вершинного шейдера, передаются как массивы (так как шейдер имеет доступ ко всем вершинам патча сразу), при этом размер массива указывать не обязательно.

```
in vec3 normal [];          // per-vertex normal
```

Что поступает на вход

gl_in — массив вершин (состоит из gl_Position, gl_PointSize и gl_ClipDistance[])

gl_PatchVerticesIn — количество вершин патча

gl_PrimitiveID — индекс примитива (патча), по которому нужно записать выходные данные. Записывать по другому индексу нельзя. Порядковый номер патча gl_PrimitiveID считается в рамках одного вызова glDraw*

gl_InvocationID — порядковый номер выходной вершины

Что поступает на выход

gl_out — массив вершин

gl_TessLevelInner — данные для тесселятора (массив из 2x float) — управляет разбиением внутренностей примитива

gl_TessLevelOuter — ещё данные для тесселятора (массив из 4x float) — управляет разбиением границ примитива, каждое число отвечает за определённую сторону

Tessellation Control Шейдер. Пример кода

```
#version 410 core
#define id gl_InvocationID  // номер вершины

uniform float inner;
uniform float outer;
layout(vertices = 3) out; // задаёт размер патча в 3 вершины from vertex shader

in VertexCS  {
  vec3 position;
  vec2 texcoord;
  vec3 normal;
} vertcs[];

// to evaluation shader
out VertexES  {
  vec3 position;
  vec2 texcoord;
  vec3 normal;
} vertes[];
```

Tessellation Control Шейдер. Пример кода

```
void main(void) {
  vertes[id].position = vertcs[id].position;
  vertes[id].texcoord = vertcs[id].texcoord;
  vertes[id].normal = vertcs[id].normal;

  if (0 == id) {
    gl_TessLevelInner[0] = inner;
    gl_TessLevelInner[1] = inner;
    gl_TessLevelOuter[0] = outer;
    gl_TessLevelOuter[1] = outer;
    gl_TessLevelOuter[2] = outer;
    gl_TessLevelOuter[3] = outer;
  }
  gl_out [id].gl_Position = gl_in[id].gl_Position;
}
```

Tessellation Control Шейдер

- Патч отбрасывается если:
  - glTessLevelOuter[x] <= 0
  - glTessLevelOuter[x] = NaN

Доступ к вершинам патча

Шейдер имеет доступ не только к входным данным для каждой вершины, но также и к выходным данным для каждой вершины патча (независимо от того для какой вершины он был вызван в данный момент).

Это позволяет при расчете одних вершин патча обращаться к результатам обработки других вершин того же патча.

Однако это несет в себе опасность, поскольку порядок вызова шейдера для обработки вершин одного и того же примитива, неопределён.

Барьерная синхронизация — barrier()

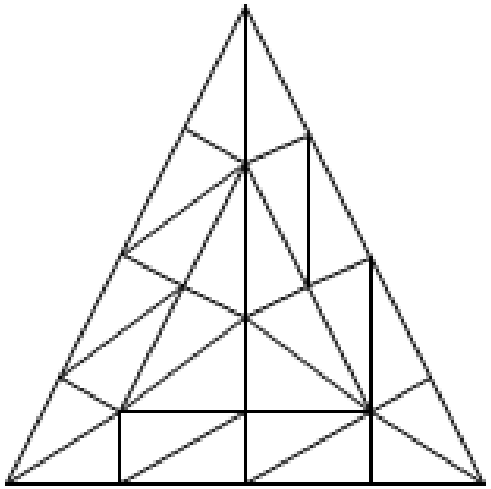
Для того, чтобы избежать опасности доступа к данным, которые еще не просчитаны, в tessellation control shader добавлена функция барьерной синхронизации — barrier().

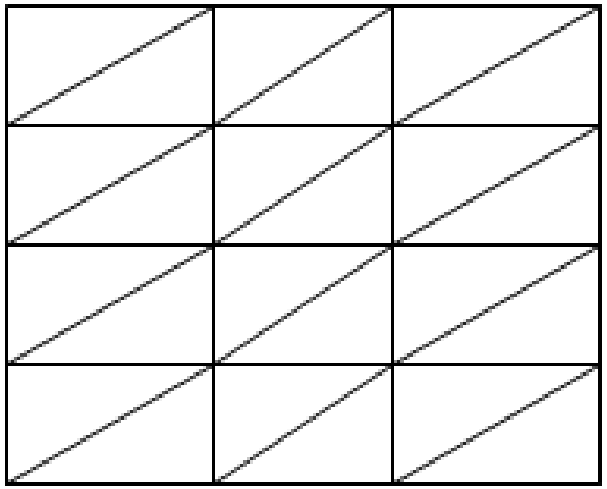
Если при обработке вершины встречается вызов этой функции, то дальнейшая обработка этой вершины приостанавливается до тех пор, пока при обработке всех остальных вершин этого патча, не будет вызвана эта функция. Только после этого обработка вершин данного патча будет продолжена.

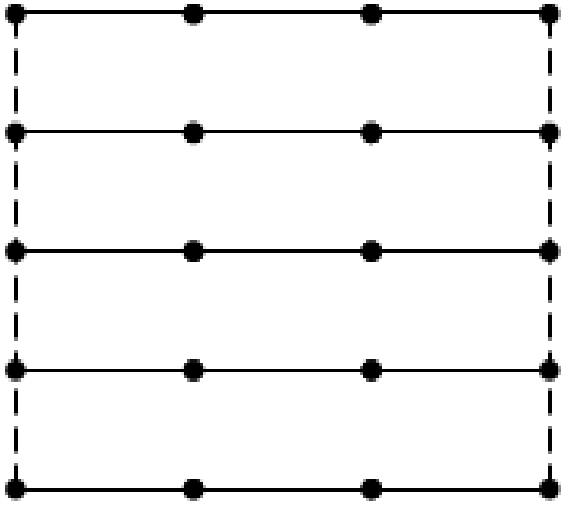
Тем самым, барьерная синхронизация гарантирует, что никогда не получится так, что обработка одних вершин находится до вызова barrier, а обработка других вершин того же патча — после вызова.

Использование функции barrier позволяет организовать безопасный доступ к выходным значениям для других вершин того же патча.

Типы тесселяции примитивов — triangles, quads и isolines



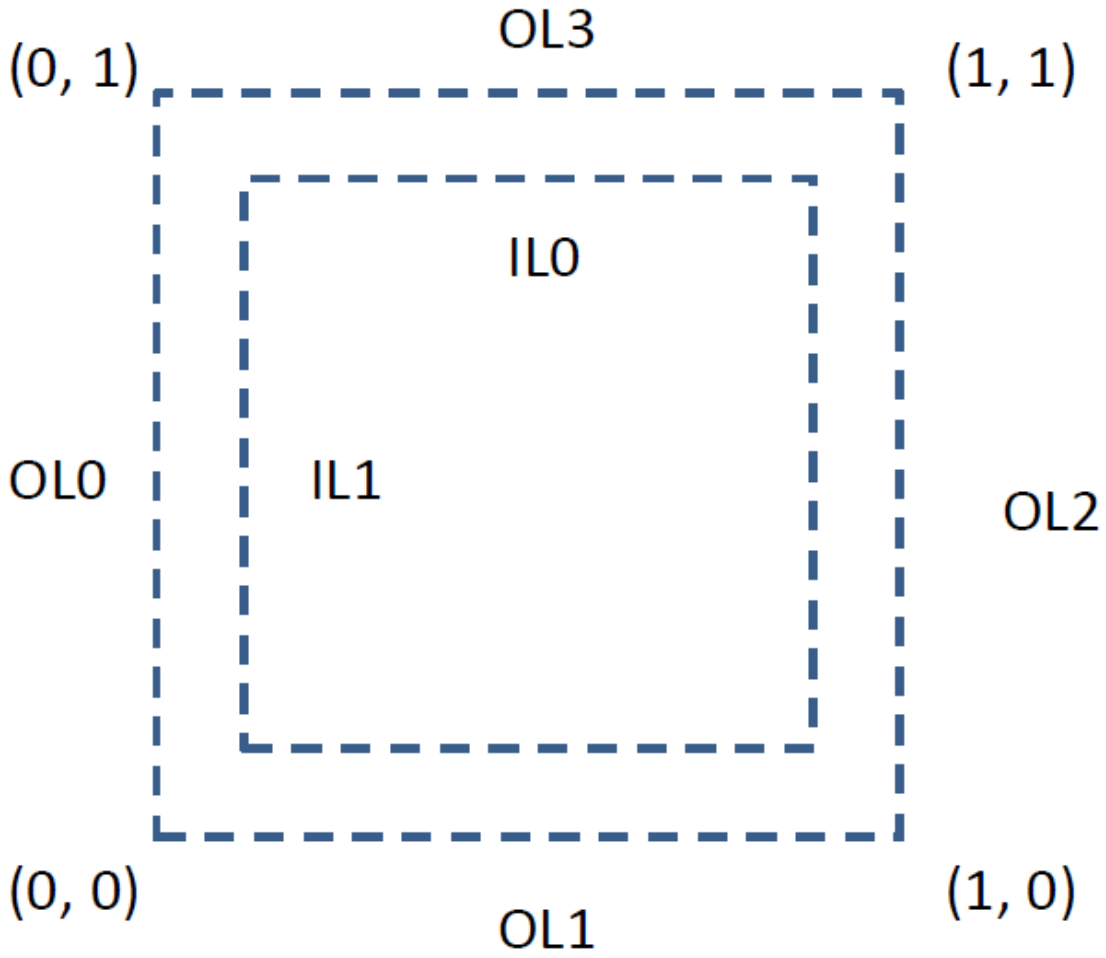


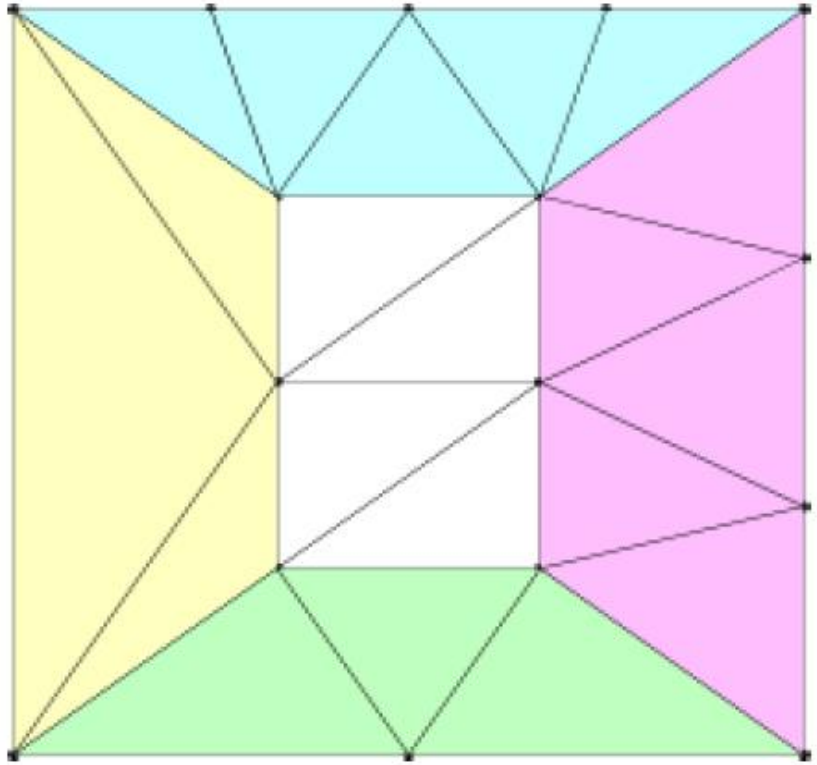


Тесселятор

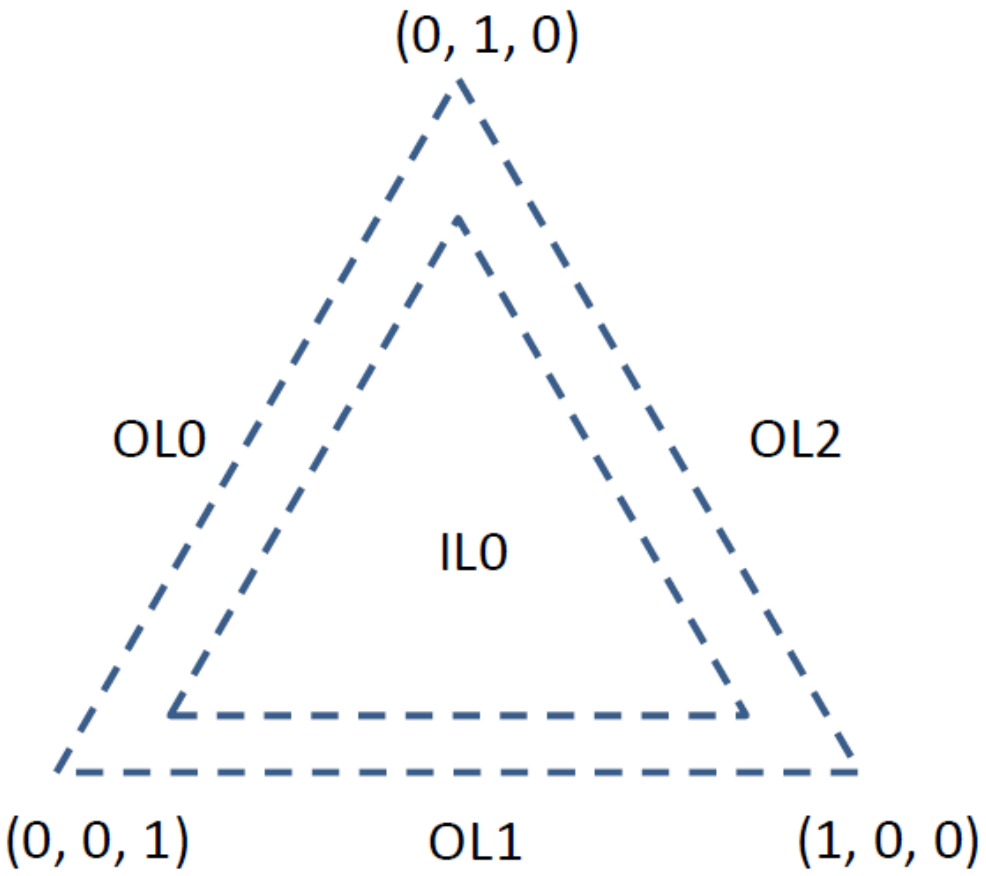
- Преобразует входной патч в новый набор основных примитивов:
  - Точки, линии, треугольники
- Использует факторы тесселяции для разбиения
- Тип разбиения задается определением layout

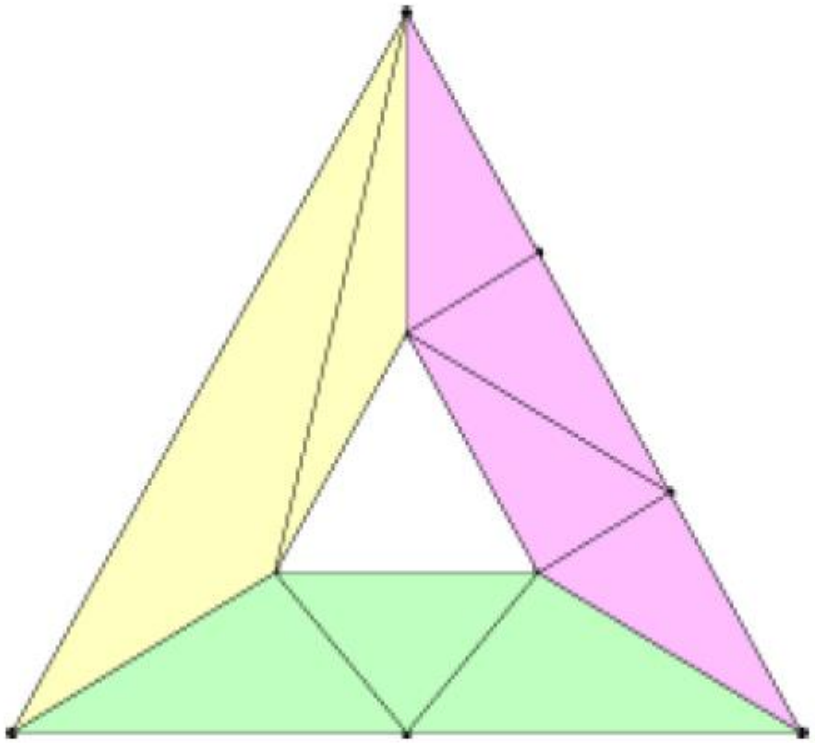
Тесселятор. Типы разбиений. Четырехугольники





Тесселятор. Типы разбиений. Треугольники





Tessellation Evaluation Shader

- Вызывается для каждой вершины, созданной тесселятором
- Обрабатывает полученные в результате тесселяции примитивы
- Принимает патч в качестве входных данных

Что поступает на вход

gl_in — массив вершин
gl_PatchVerticesIn — количество вершин ИСХОДНОГО патча
gl_PrimitiveID — индекс примитива
gl_TessCoord — позиция вершины в патче (x, y, z)
gl_TessLevelInner — массив внутренних уровней тесселяции
gl_TessLevelOuter — массив внешних уровней тесселяции

Что на выход

На выходе у него одна вершина (gl_Position, gl_PointSize и gl_ClipDistance[])

Tessellation Evaluation Шейдер. Входные/выходные данные

```
#version 410 core
// указывает способ разбиения треугольников - разбиение ребер на равные части
layout(triangles, equal_spacing) in;

uniform mat4 modelViewProjectionMatrix;
uniform sampler2D heightmap;

in VertexES { // from control shader
  vec3 position;
  vec2 texcoord;
  vec3 normal;
}
vertes[];

out VertexFS { // to fragment shader
  vec3 position;
  vec2 texcoord;
  vec3 normal;
} vertfs;
```

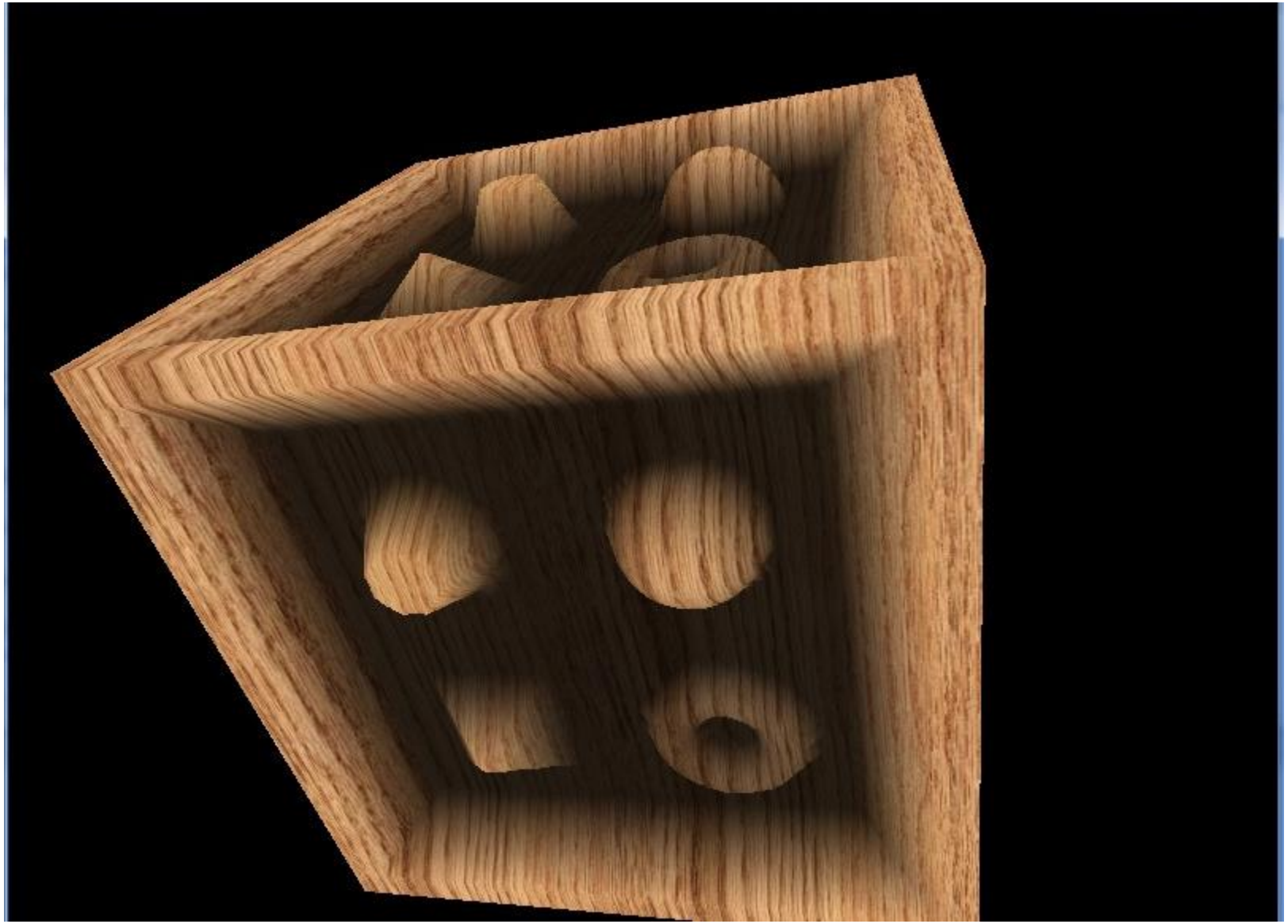
Вычисление данных для передачи во фрагментный шейдер

Из control shader`а нам пришли данные о исходном патче (in VertexES), на основе которых мы должны вычислить их же, но для текущей вершины. Делается это простой интерполяцией:

```
vec2 interpolate2D(vec2 v0, vec2 v1, vec2 v2) {
  return vec2(gl_TessCoord.x) * v0 + vec2(gl_TessCoord.y) * v1 + vec2(gl_TessCoord.z) * v2;
}

vec3 interpolate3D(vec3 v0, vec3 v1, vec3 v2) {
  return vec3(gl_TessCoord.x) * v0 + vec3(gl_TessCoord.y) * v1 + vec3(gl_TessCoord.z) * v2;
}

void main(void) {
  vertfs.position = interpolate3D(vertes[0].position, vertes[1].position, vertes[2].position);
  vertfs.texcoord = interpolate2D(vertes[0].texcoord, vertes[1].texcoord, vertes[2].texcoord);
  vertfs.normal = normalize(interpolate3D(vertes[0].normal, vertes[1].normal, vertes[2].normal));

…
```

Вычисляем позицию вершины, смещаем её и переводим в screen space

```
…
    gl_Position = modelViewProjectionMatrix * (
        vec4(interpolate3D(gl_in[0].gl_Position.xyz, gl_in[1].gl_Position.xyz, gl_in[2].gl_Position.xyz), 1.0)
        - vec4(vertfs.normal, 1) * (texture(heightmap, vertfs.texcoord) / 4.0) );
}
```



Тесселяционные схемы

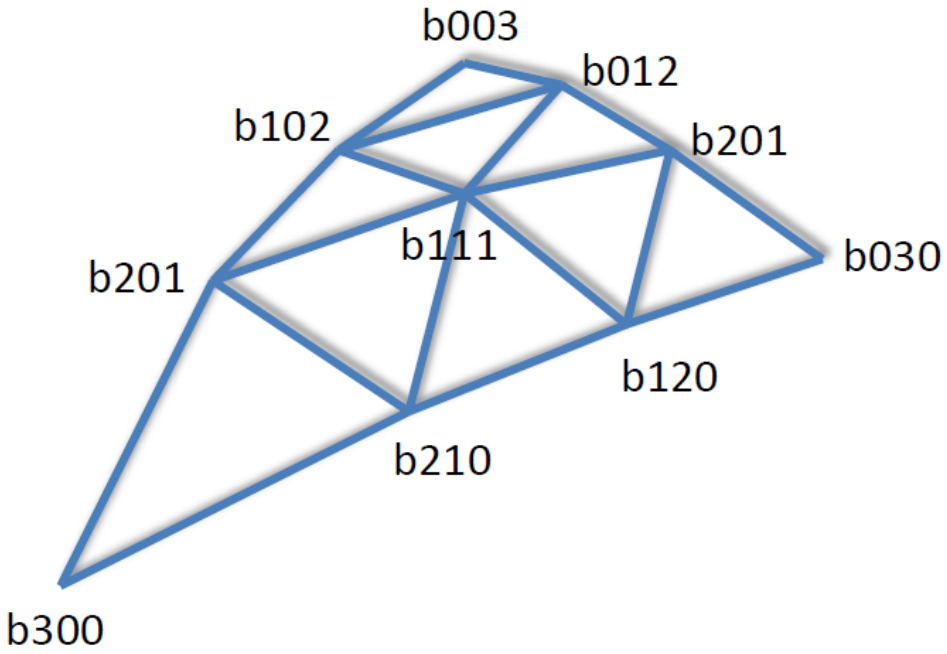
- Плоская тесселяция (дайсинг)
- PN-Треугольники
- Фонг-Тесселяция
- Патчи

- Использование карт глубины для создания рельефа

PN-Треугольники

Построение по данным отдельного треугольника кубического патча Безье

$$p(s,t,u) = (\alpha s + \beta t + \gamma u)^3 = \beta^3 t^3 + 3\alpha\beta^2 st^2 + 3\beta^2\gamma t^2 u + 3\alpha^2\beta s^2 t + 6\alpha\beta\gamma stu + 3\beta\gamma^2 tu^2 + \alpha^3 s^3 + 3\alpha^2\gamma s^2 u + 3\alpha\gamma^2 su^2 + \gamma^3 u$$

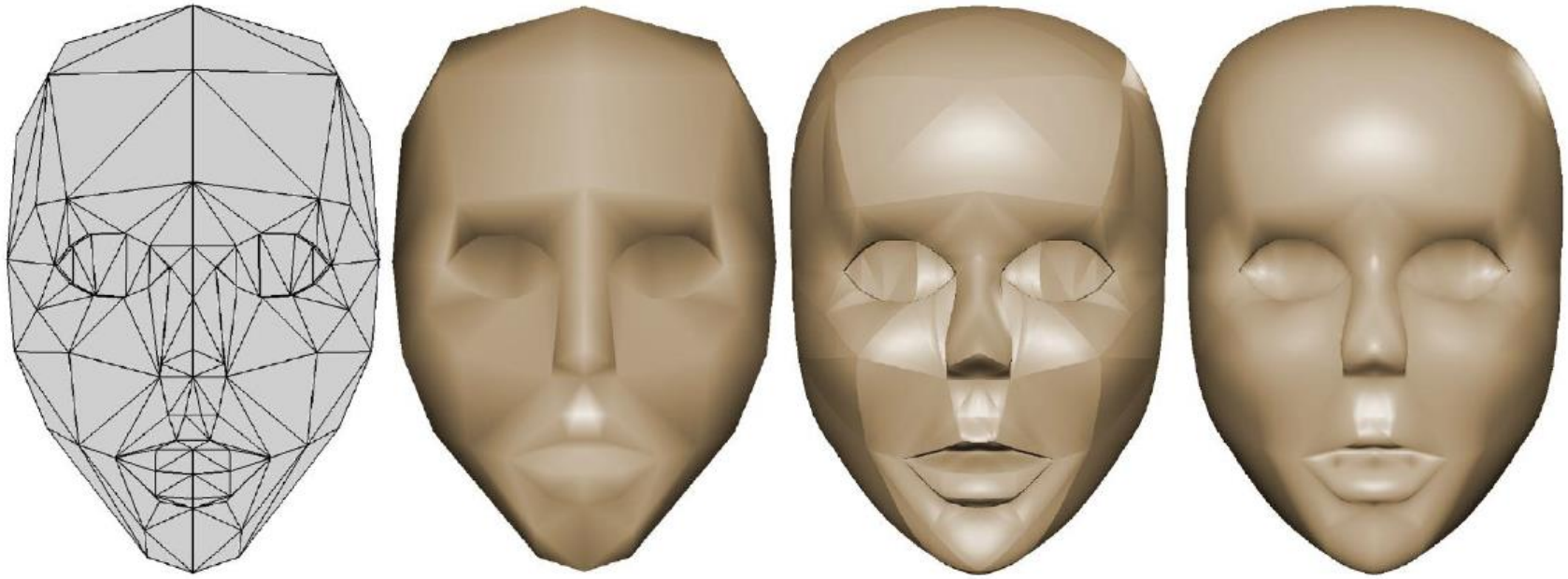


PN-Треугольники

$$b_{111} = E * 3/2 + V * 1/2$$

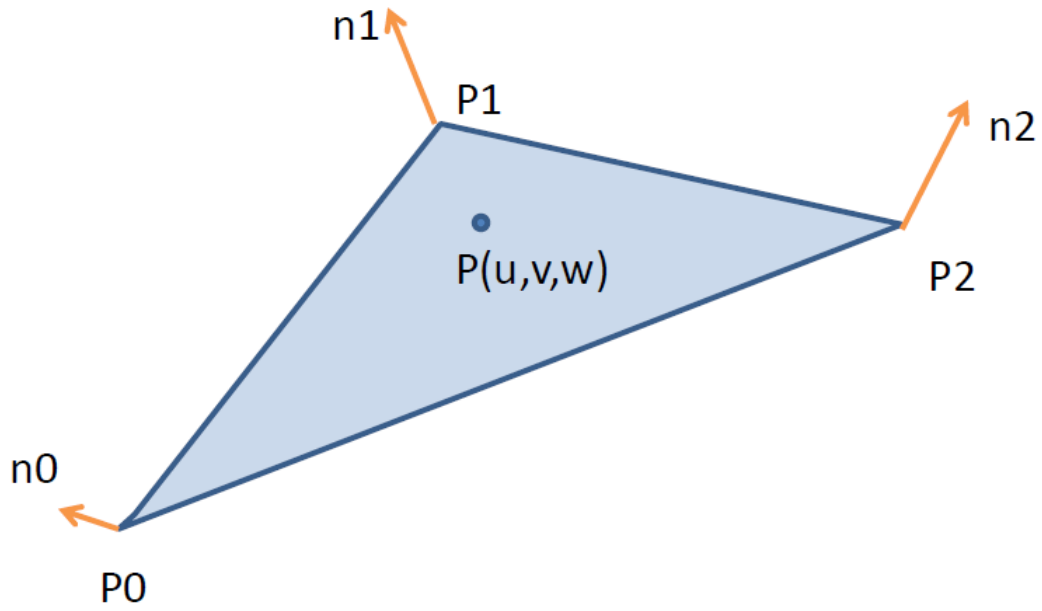
$$E = (b_{210} + b_{120} + b_{012} + b_{012} + b_{102} + b_{201}) / 6$$

$$V = (b_{003} + b_{030} + b_{300}) / 3$$



Фонг-Тесселяция

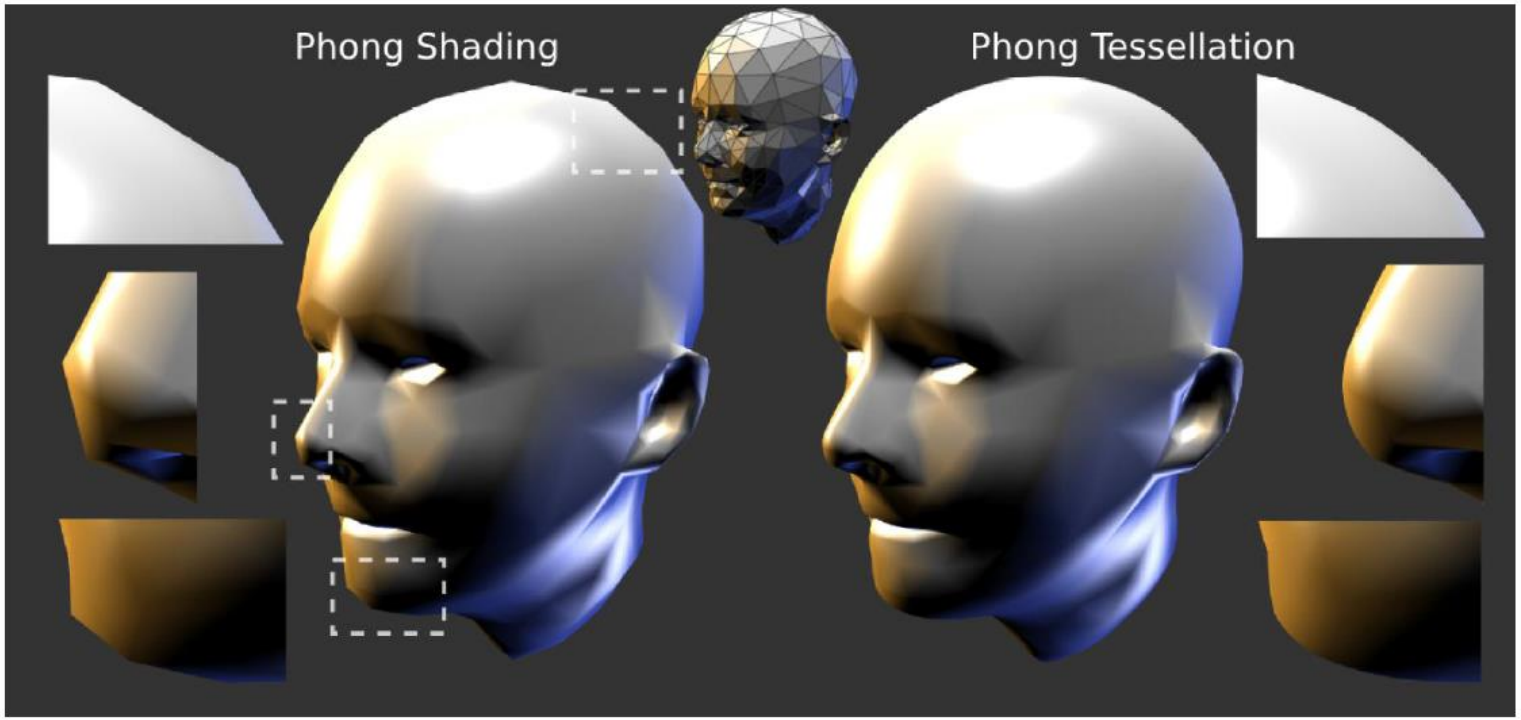
- Есть три плоскости, заданные парами (P_i, n_i)
- Pi* - проекции P на эти плоскости
- $P_{phong} = P^*_0 * u + P^*_1 * v + P^*_2 * w$

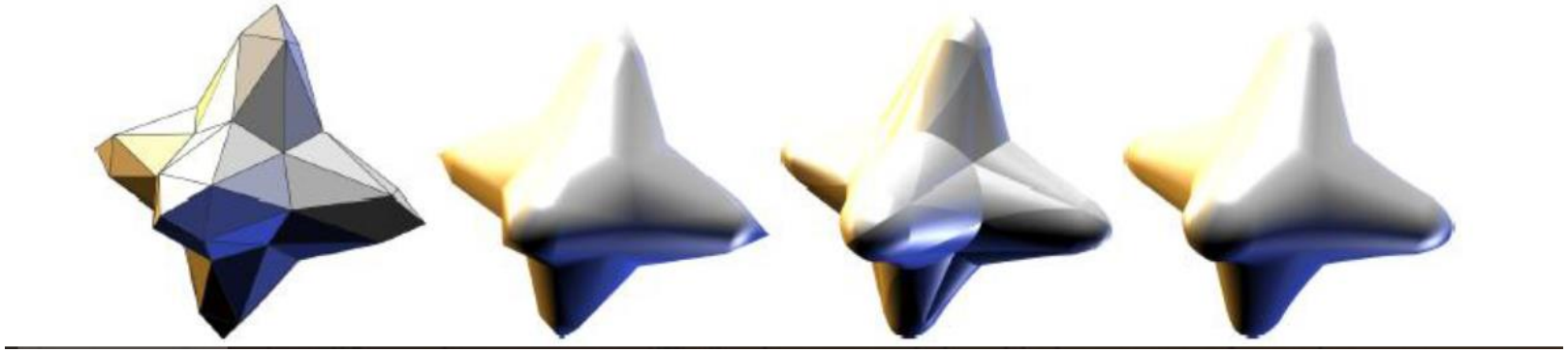


Области применения

- Ландшафты
- Повышение уровня детализации геометрических моделей
- Рендеринг травы, волос, частиц

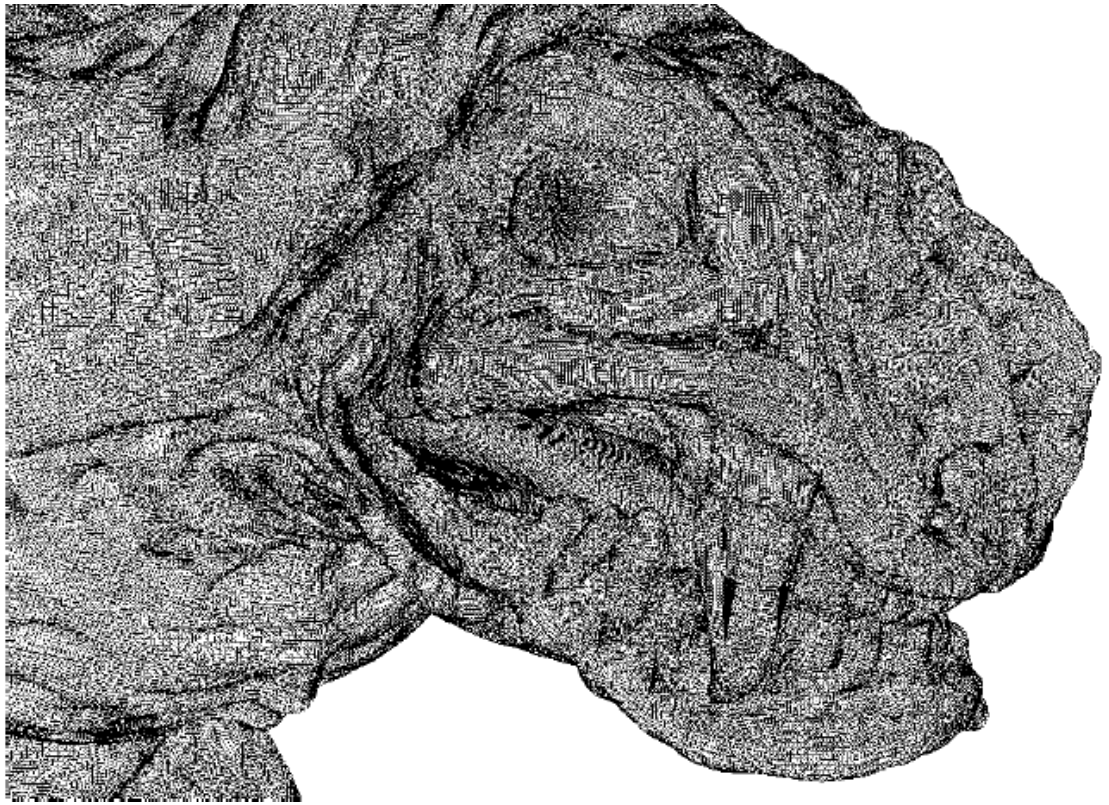
Фонг-Тесселяция





Эвристики для разбиения

- Разбиение по расстоянию
- По ориентации патча
- Адаптивное разбиение в экранном пространстве



Оптимизации

- Уменьшение числа атрибутов

- Отбрасывание невидимых патчей
  - По видимому объему
  - С учетом взаимного перекрытия