Тесселяция

Компьютерная графика

План

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

Проблема детализации геометрии

Как эффективно отрисовывать сложные гладкие поверхности (персонажи, рельеф, органические объекты), не создавая для них гигантские статические сетки полигонов?

Проблемы статических сеток:

- Перегруженность: Объект всегда отрисовывается с максимальной детализацией, даже если он далеко от камеры. Это трата вычислительных ресурсов (вершинный шейдер, пропускная способность памяти).
- Недостаточность: Приближаясь к объекту, видны грубые полигоны.
- Жесткость: Уровень детализации (LOD) должен быть подготовлен художником заранее в нескольких вариантах.

Решение:

Программируемая тесселяция. Позволяет динамически, на GPU, разбивать простые патчи (заготовки) на нужное количество треугольников непосредственно перед растеризацией.

Мотивация

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

Base model

Bump mapping

Displacement mapping

Что это такое?

Тесселяция — это разбиение некоторого примитива на несколько меньших.

Тесселяционный конвейер в OpenGL 4.0

Аналогия для понимания:

TCS (**Контрольный шейдер**) -> Архитектор. Принимает «набросок» (патч). Решает, на сколько частей его нужно разбить по разным осям (**задаёт уровни тесселяции**). Может немного подправить контрольные точки.

TPG (**Генератор примитивов**) -> Конвейерный рабочий. Слепо выполняет план архитектора. Берёт патч и **создаёт** согласно указаниям **множество новых вершин**, но не зная, где им точно находиться в пространстве. Он **знает** только их **координаты внутри** исходного **патча** (**UV-координаты**, или **gl_TessCoord**).

TES (Шейдер оценки) -> Дизайнер-декоратор. Получает от рабочего новую вершину и её «адрес» внутри патча (**gl_TessCoord**). **Имея** исходный «набросок» (контрольные точки), **вычисляет окончательную 3D-позицию** этой вершины по заданной формуле (например, по уравнению кривой **Безье** или поверхности **NURBS**).

Ключевые понятия

Патч (Patch): **Примитив**, который является **исходными** данными **для тесселяции**. Определяется **набором контрольных точек** (control points). Количество точек на патч задаётся (от 1 до 32).
Включает в себя как вершинные атрибуты, так и атрибуты задаваемые для всего патча.

Контрольные точки (Control Points): **Вершины**, описывающие **исходную форму**. Обычно их 3 (треугольный патч), 4 (квадратный/квад патч), 16 (бикубическая поверхность Безье). Проходят через вершинный шейдер и становятся **gl_in[] в TCS**.

Уровни тесселяции (Tessellation Levels): Главный параметр, управляющий детализацией.

- Внешние уровни (Outer Levels - gl_TessLevelOuter[4]): Определяют, на сколько сегментов разбивается каждая внешняя сторона патча.
- Внутренний уровень (Inner Levels - gl_TessLevelInner[2]): Определяет разбиение внутренней области патча.
- Задаются в TCS и используются TPG.

Tessellation Control Шейдер. Подробнее

- Это программируемый шейдер (как вершинный или фрагментный). Вы пишете для него код на GLSL.
- 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)

Задачи:

Записать уровни тесселяции в gl_TessLevelOuter и gl_TessLevelInner.

При необходимости, преобразовать/скопировать контрольные точки из входного массива gl_in[] в выходной gl_out[]. Часто это просто копирование.

Важно: Все инварианты для одного патча (особенно уровни тесселяции) должны вычисляться одинаково во всех вызовах TCS для этого патча!

TCS. Инстансы

TCS Работает группами (инстансами). Это его самая важная особенность.

gl_InvocationID — это встроенная переменная, номер текущего вызова шейдера внутри группы.

Одна группа = Один патч

Внутри группы шейдер вызывается один раз для каждой контрольной точки этого патча.

Пример: У вас патч из 4 точек (квадрат). Значит, для отрисовки одного такого патча TCS запустится 4 раза (инстанса). gl_InvocationID будет принимать значения 0, 1, 2, 3. Каждый инстанс обрабатывает одну конкретную контрольную точку.

Патч. Установка параметров

Число вершин задаётся
или в **tesselation control shader**
При помощи директивы layout в нём явно задается число вершин в примитиве.
layout(vertices = 3) out; // N - количество контрольных точек на выходе (и обычно на входе)

или, если этот шейдер отсутствует, в коде самого приложения, при помощи команды
glPatchParameteri(GL_PATCH_VERTICES, vertexCount)

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

Параметры для glDrawArrays

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);

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

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 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 460 core
layout(vertices = 4) out; // 4 контрольные точки на выходе

void main() {
    // Копируем контрольные точки
    gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;

    // Задаем уровни тесселяции (здесь - статические, но можно вычислять на основе расстояния до камеры)
    if (gl_InvocationID == 0) { // Делаем это только в первом вызове для патча
        gl_TessLevelOuter[0] = 2.0; // Левая грань
        gl_TessLevelOuter[1] = 4.0; // Нижняя грань
        gl_TessLevelOuter[2] = 2.0; // Правая грань
        gl_TessLevelOuter[3] = 4.0; // Верхняя грань
        gl_TessLevelInner[0] = 4.0; // Внутреннее разбиение по U
        gl_TessLevelInner[1] = 4.0; // Внутреннее разбиение по V
    }
}
```

TCS. Ещё 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[];
```

TCS. Ещё пример кода. Продолжение

```
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

Не только копирование

Каждый инстанс TCS имеет доступ ко всем контрольным точкам патча на входе (gl_in[]) и должен заполнить соответствующую точку на выходе (gl_out[]).

gl_in[gl_InvocationID].gl_Position — позиция своей точки на входе (из вершинного шейдера).
gl_out[gl_InvocationID].gl_Position — позиция своей точки на выходе (пойдет в TES).

Чаще всего это просто копирование

Но здесь можно делать полезные вещи:

- Сглаживание/схлопывание патча: Изменять позиции контрольных точек, чтобы повлиять на итоговую поверхность.
- Анимация контрольной сетки: Двигать точки перед тесселяцией.
- Переиндексация: Скопировать данные не один-в-один, а по другой схеме.

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()

Проблема: Инстансы TCS выполняются параллельно и независимо. Что если инстанс 0 читает данные, которые должен был записать инстанс 1? Без синхронизации это **data race (состояние гонки)**.

Решение: Вызов функции **barrier()**.

Это команда «СТОП!» для всех инстансов в патче.

Все инстансы ждут, пока все не дойдут до этой точки в коде.

Только после этого выполнение продолжается.

Когда использовать? Если инстансы обмениваются данными через gl_out[].

Пример: Инстанс 0 хочет прочитать измененную инстансом 1 точку. Инстанс 1 должен её записать, потом оба должны вызвать barrier(), и только потом инстанс 0 может ее читать.

Стандартный порядок команд в TCS:

- Копируем/обрабатываем свою контрольную точку (gl_out[ID] = ...)
- barrier(); // Ждем, чтобы все точки были готовы

- (Опционально) Читаем точки соседей для расчетов

- В инстансе с ID == 0 вычисляем и записываем gl_TessLevel*

Генератор примитивов тесселяции (TPG) - фиксированный этап

Не программируется. На основании уровней от TCS генерирует сетку новых вершин

Преобразует входной патч в новый набор основных примитивов: точки, линии, треугольники

Выход:
Для каждой новой вершины определяет её нормализованные координаты внутри патча -
gl_TessCoord (vec3)

Для квадратного (quad) патча: gl_TessCoord.xy - это UV-координаты в диапазоне [0, 1]

Для треугольного (tri) патча: gl_TessCoord.xyz - это барицентрические координаты

Шейдер оценки тесселяции (TES)

Вызывается на каждую вершину, сгенерированную TPG

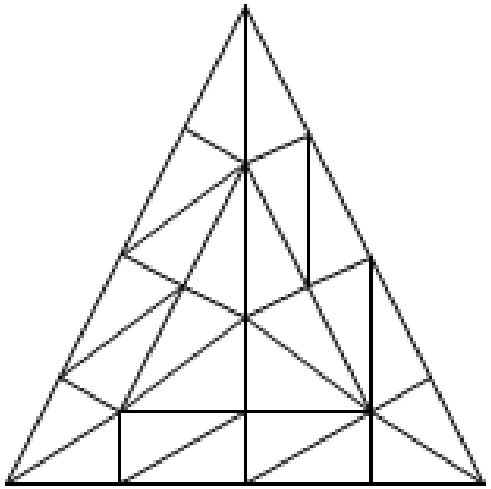
Получает на вход:
- gl_TessCoord — адрес вершины внутри патча
- Контрольные точки из gl_in[] (те, что вышли из TCS)

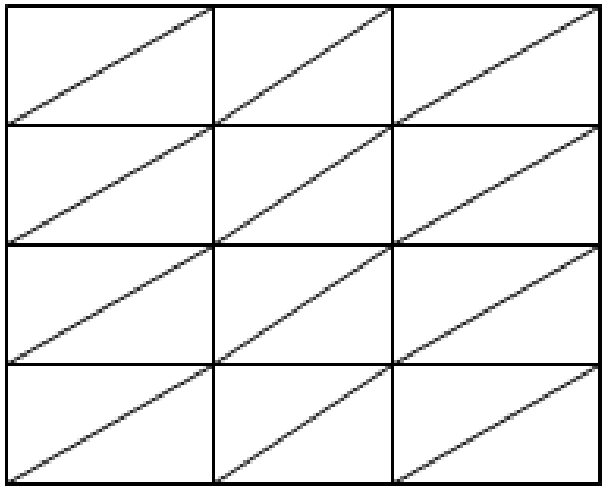
Главная задача:
Используя gl_TessCoord и массив контрольных точек, вычислить итоговую позицию gl_Position

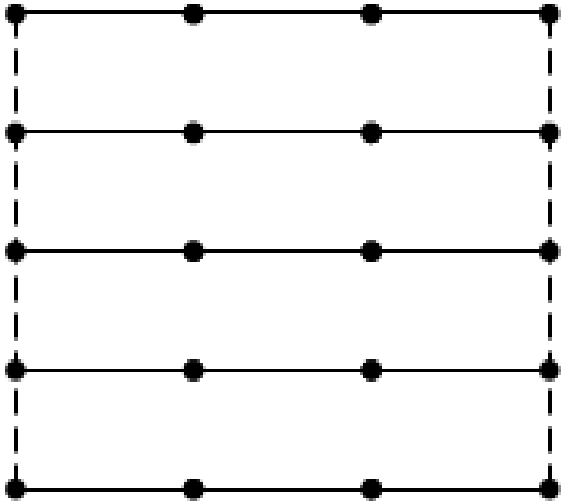
Здесь **реализуется математика интерполяции**:
- билинейная,
- Безье,
- Кунса,
- NURBS
- и т.д.

Также можно вычислять нормали, текстурные координаты для новой вершины

Типы тесселяции примитивов — triangles, quads и isolines







layout

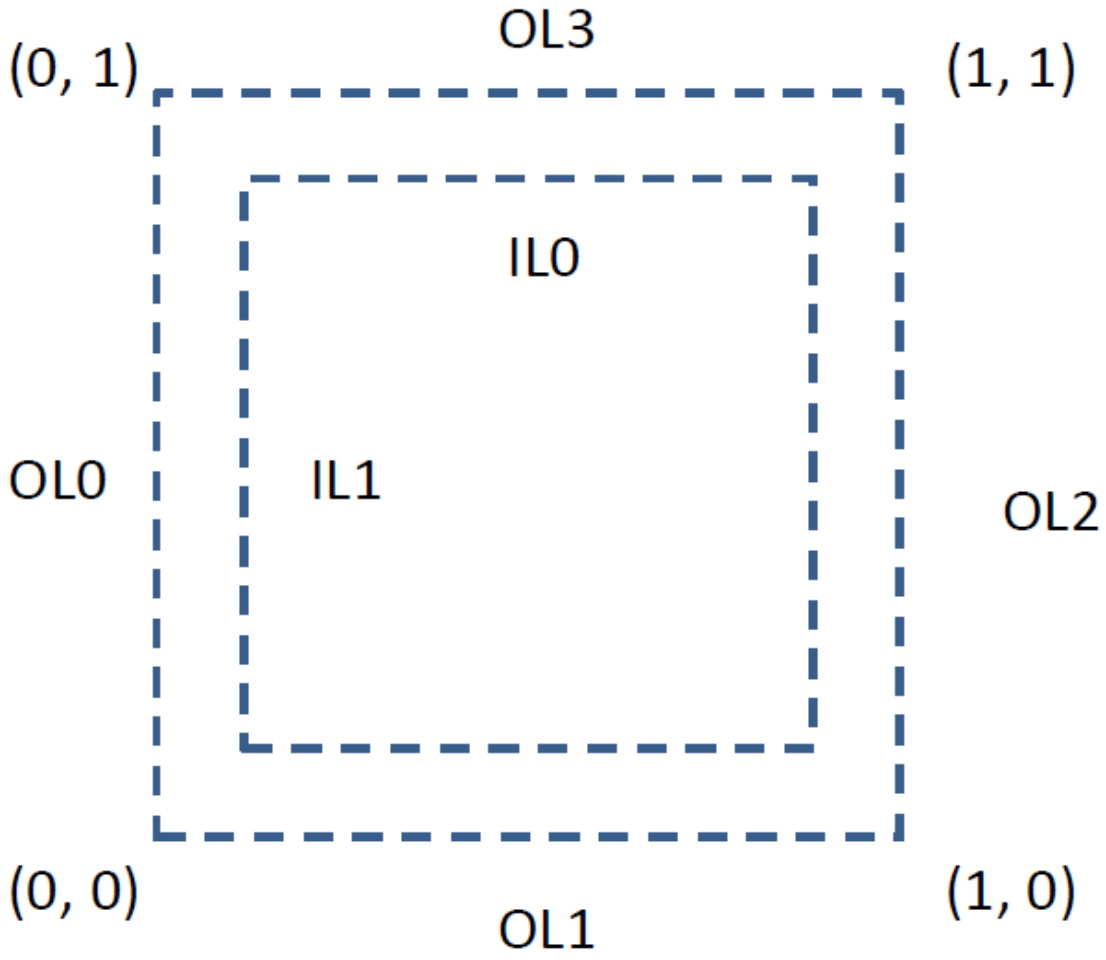
Тип разбиения задается определением layout

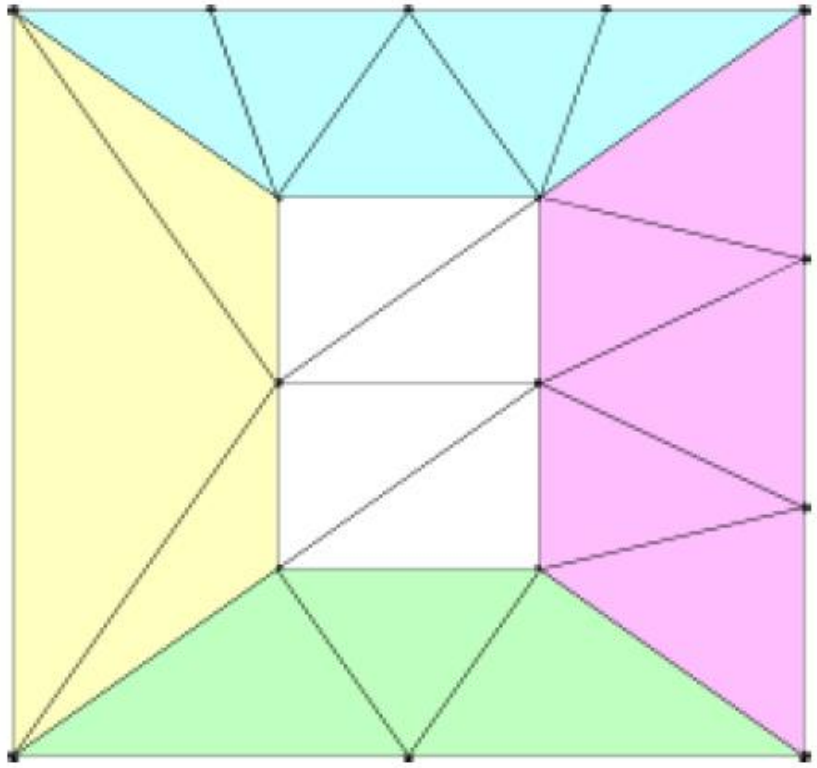
- layout(тип входного домена, разбиение, топология)
  - triangles, quads, isolines
  - equal_spacing, fractional_even_spacing, fractional_odd_spacing
  - cc, ccw, point_mode

- layout(triangles, equal_spacing, ccw)

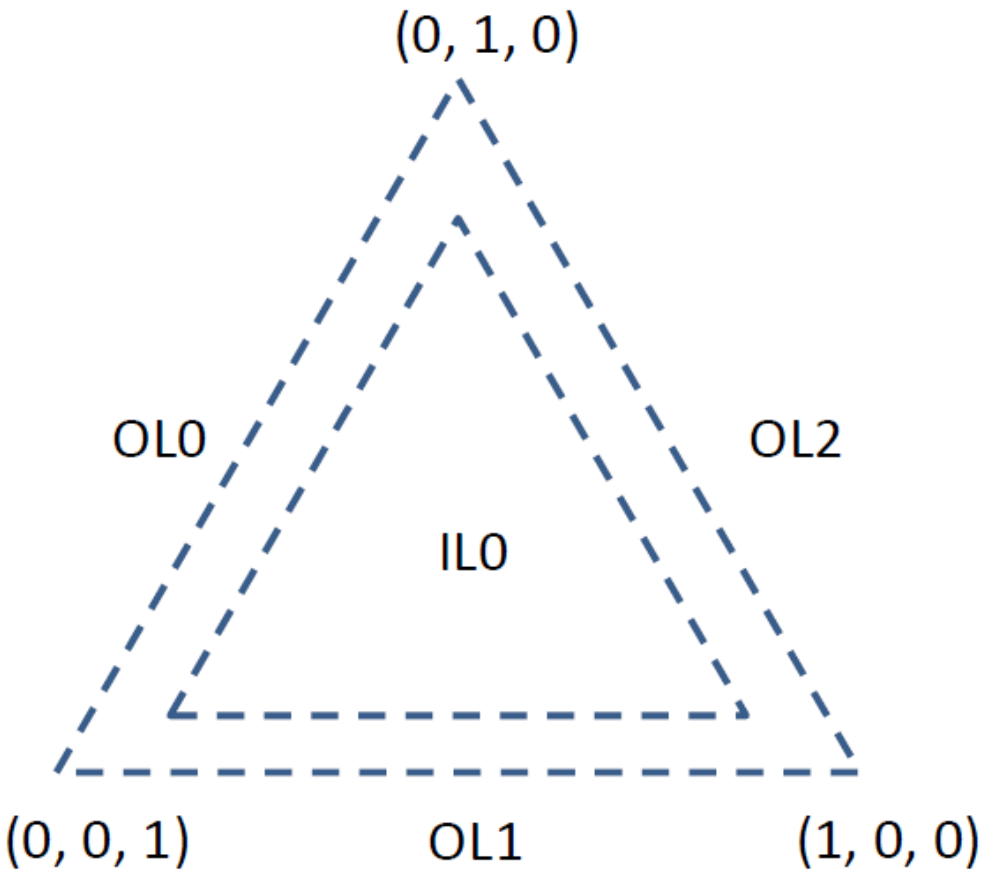
Параметр	Опция	Когда использовать
Домен	quads	Квадратные поверхности, террейны, патчи Безье 4x4.
	triangles	Треугольные поверхности, сферы из треугольников, адаптивная детализация.
	isolines	Отрисовка кривых, проволочные каркасы.
Распределение	equal_spacing	Производительность, статическая детализация, где LOD не важен.
	fractional_even_spacing	В 95% случаев. Плавный, красивый LOD для динамической тесселяции.
	fractional_odd_spacing	Если нужна всегда центральная линия/вершина.
Ориентация	ccw	Стандарт OpenGL. Лицевая сторона — против часовой стрелки.
	cw	Если по какой-то причине нужно инвертировать (редко).

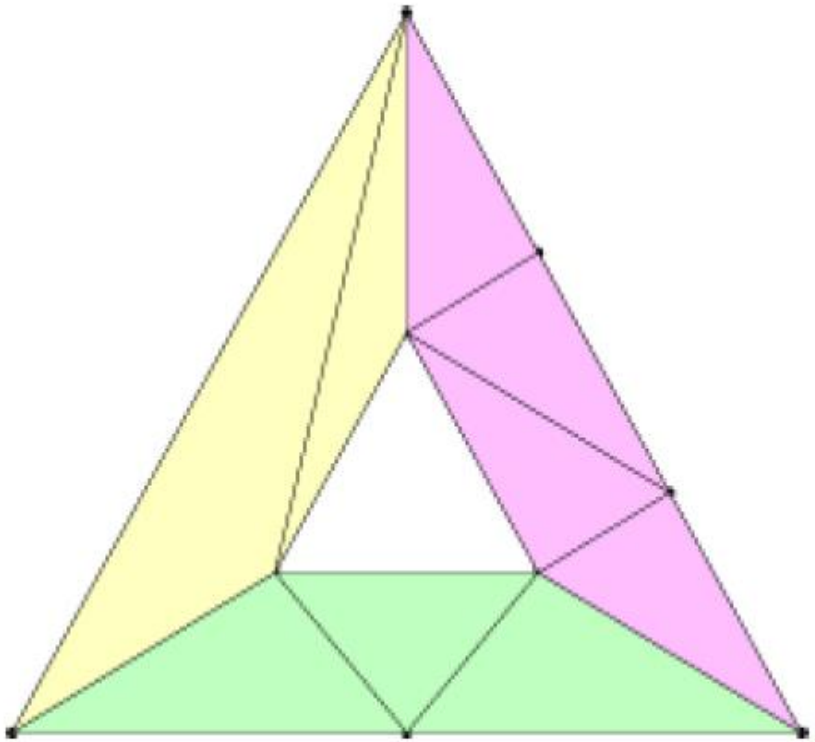
Тесселятор. Типы разбиений. Четырехугольники





Тесселятор. Типы разбиений. Треугольники





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[])

Пример TES для билинейной интерполяции квадратного патча

```
#version 460 core
layout(quads, equal_spacing, ccw) in; // Тип патча, способ разбиения, ориентация

void main() {
    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;

    // Билинейная интерполяция между 4 контрольными точками
    vec4 p0 = mix(gl_in[0].gl_Position, gl_in[1].gl_Position, u);
    vec4 p1 = mix(gl_in[2].gl_Position, gl_in[3].gl_Position, u);
    gl_Position = mix(p0, p1, v);

    // Здесь можно вычислить нормаль, если поверхность будет искривленной
}
```

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;
```

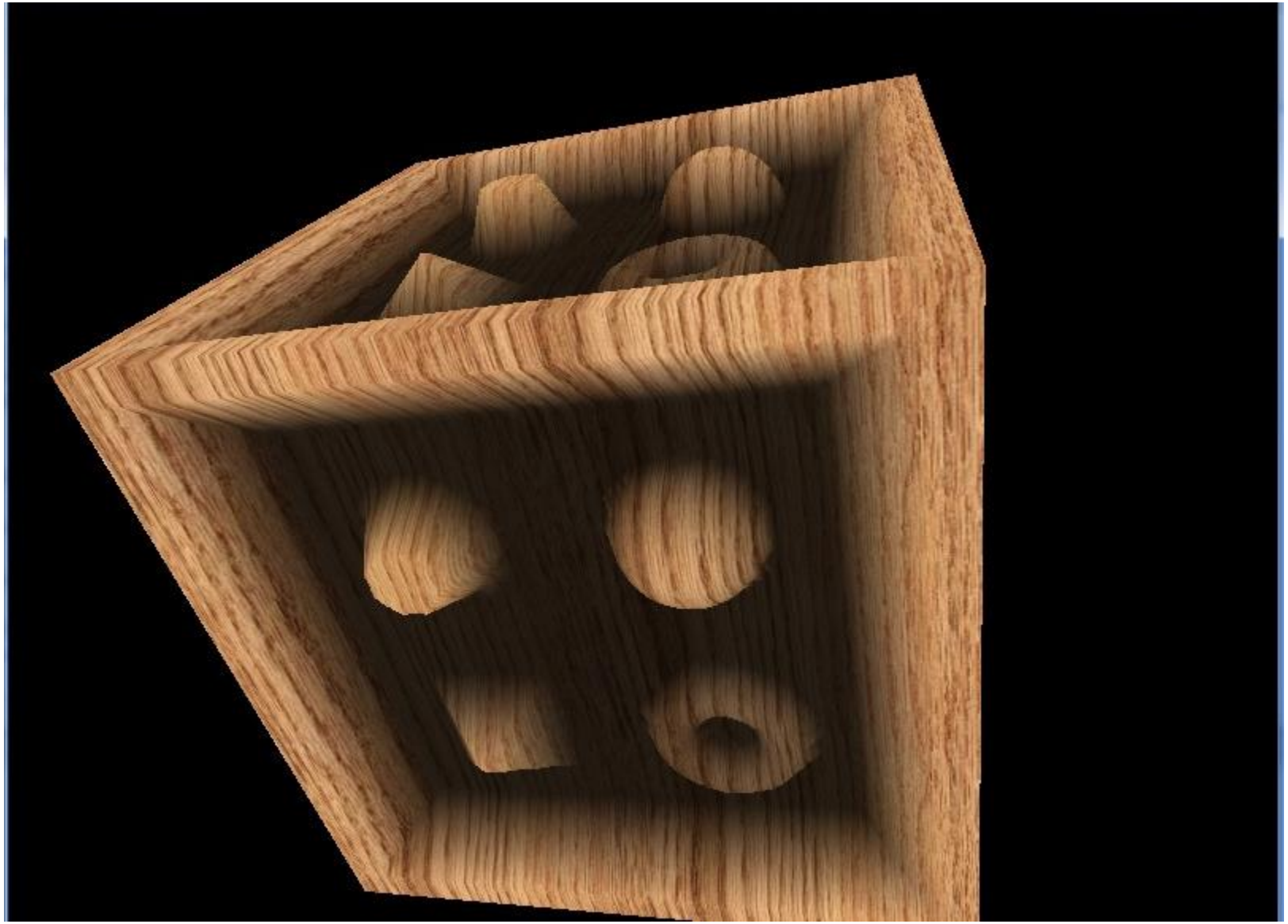
Вычисление данных для передачи во фрагментный шейдер

Из control shader`a 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

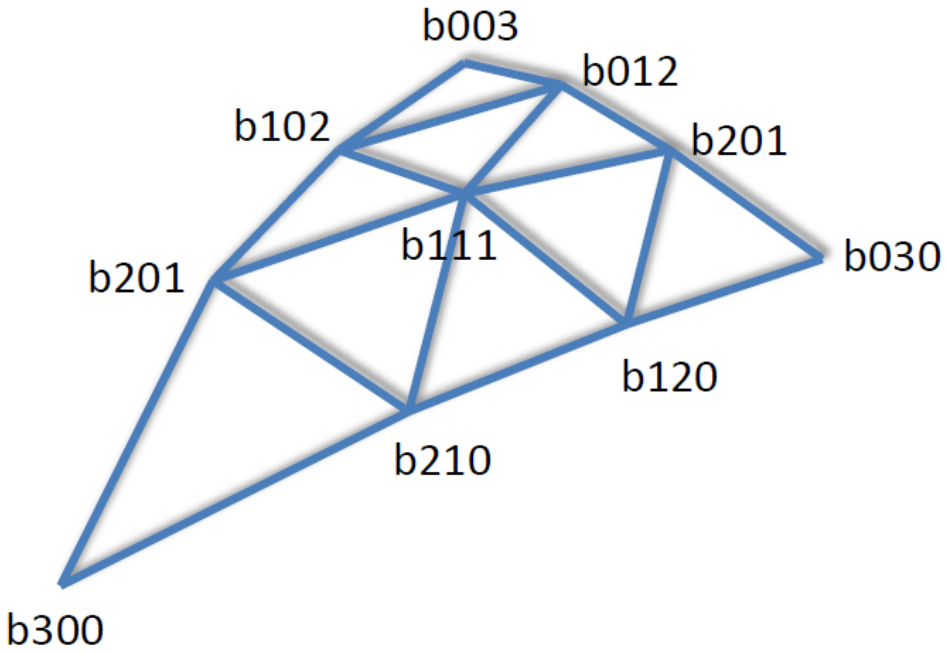
- Плоская тесселяция (дайсинг)
- PN-Треугольники
- Фонг-Тесселяция
- Патчи

- Использование карт глубины для создания рельефа

PN-Треугольники

Построение по данным отдельного треугольника кубического патча Безье

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

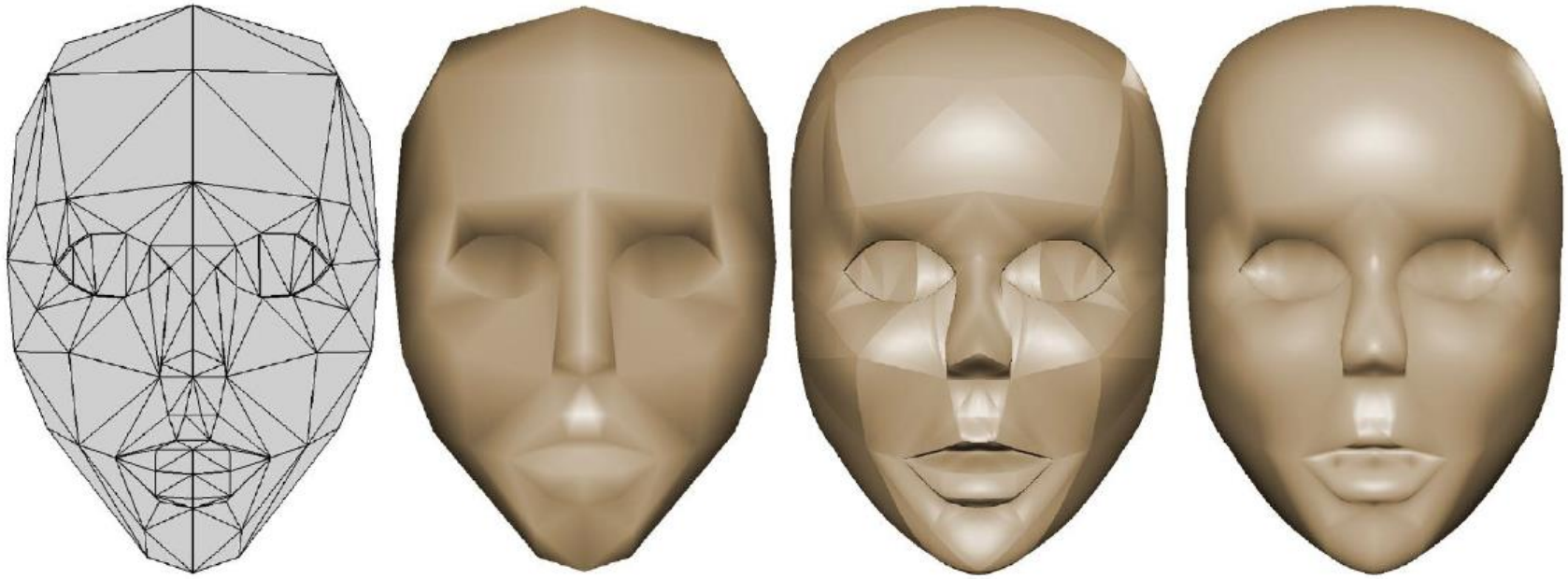


PN-Треугольники

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

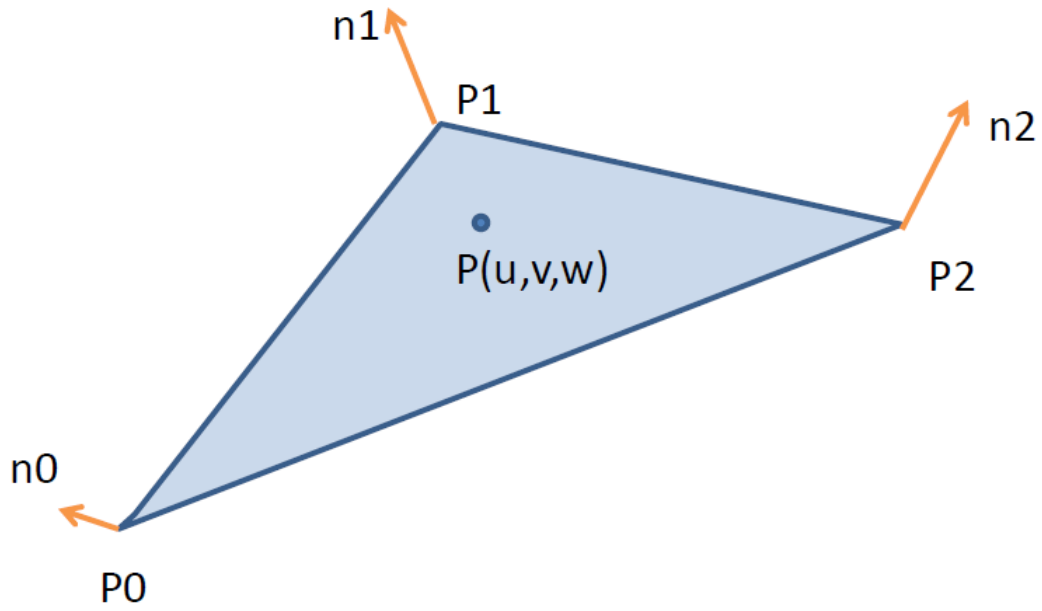
$$E = (b_{210} + b_{120} + b_{012} + b_{012} + b_{102} + b_{201}) / 6$$

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



Фонг-Тесселяция

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



Практическое применение и LOD (Уровень Детализации)

Динамическая тесселяция:

Уровни gl_TessLevel* в TCS можно вычислять на основе расстояния от патча до камеры — distance()

- Чем ближе камера -> выше уровень тесселяции -> больше треугольников
- Чем дальше камера -> ниже уровень тесселяции -> меньше треугольников

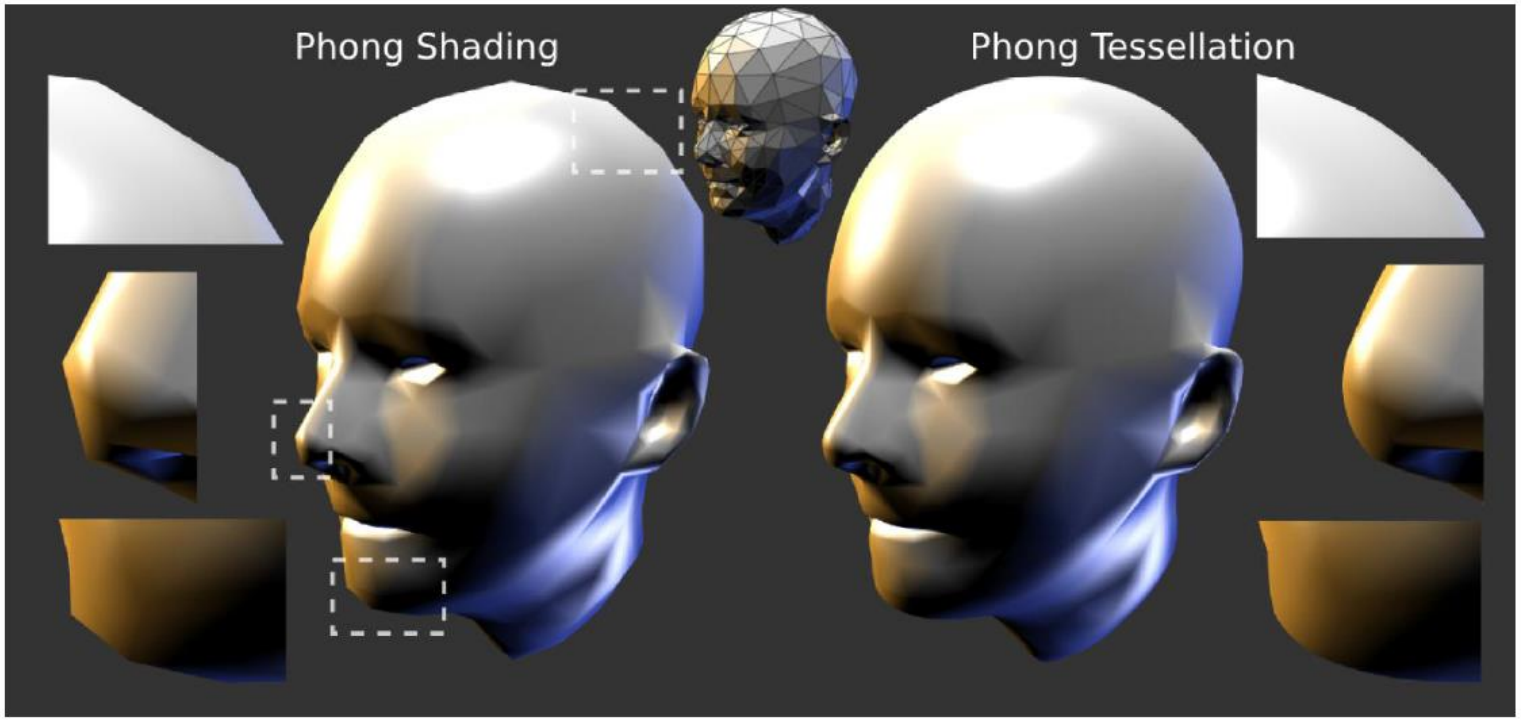
Дисплейсмент-маппинг (Displacement Mapping): Классическое применение

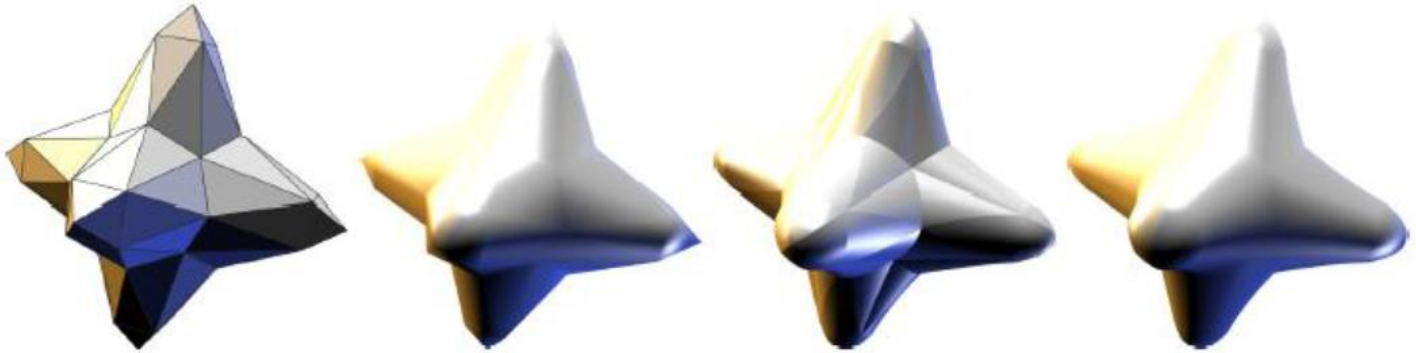
- Патч - простой квадрат (plane)
- В TES на основе gl_TessCoord читаем высоту из текстуры (карты высот)
- Смещаем вершину по нормали на эту высоту
- Результат: простая низкополигональная модель превращается в сложный каменистый рельеф только там, где это нужно

Области применения

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

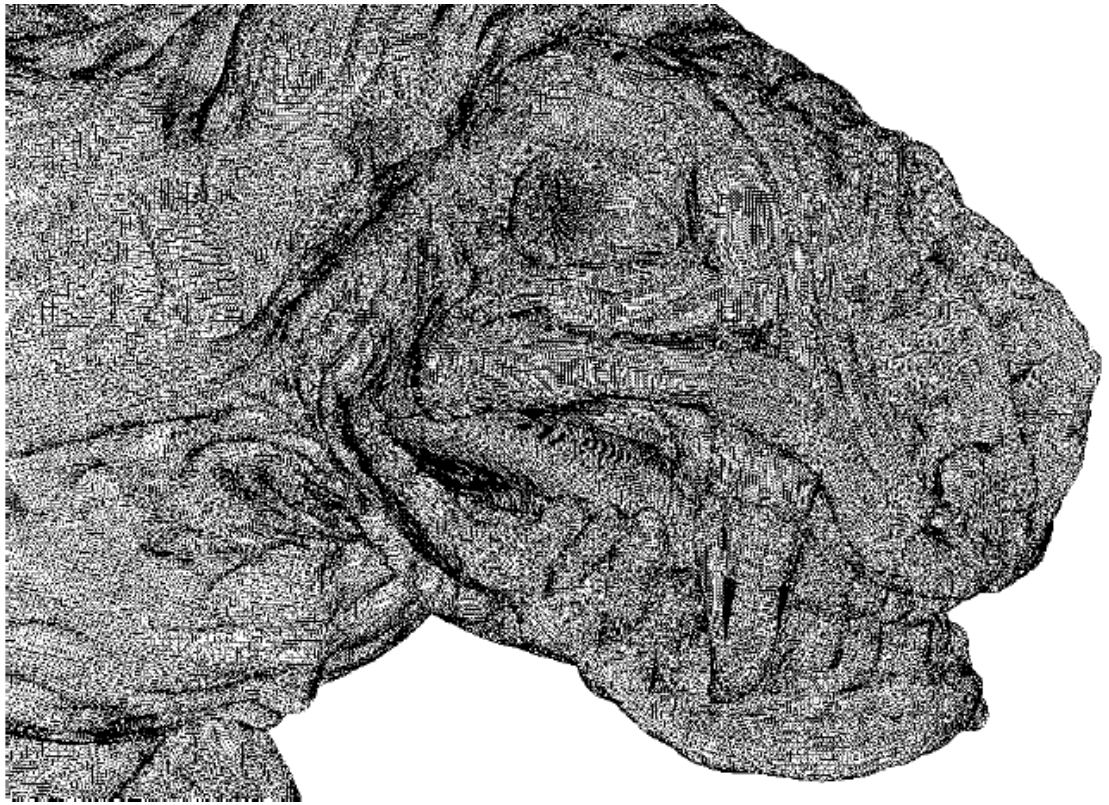
Фонг-Тесселяция





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

Резюме и преимущества

Преимущества:

- Эффективность: Геометрия создается «на лету» только необходимой детализации
- Гибкость: Один низкополигональный патч может представлять множество форм
- Уменьшение нагрузки на CPU и память: Основная работа на GPU, не нужно хранить/передавать много вершин

Сложности:

Усложняется конвейер и отладка

Нужно аккуратно выбирать уровни тесселяции, чтобы избежать «прыгающей» геометрии (трещин) или слишком резких изменений