

**Текстуры.
Удаление невидимых поверхностей.
Композиты**

Алексей Викторович Игнатенко

Лекция 10

17 ноября 2011



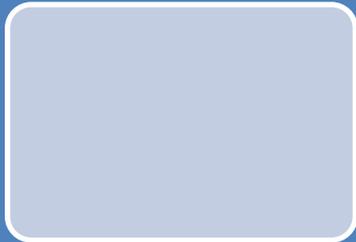
Три основных темы, все связаны с этапом растеризации



Текстурирование



Удаление невидимых поверхностей



Композиты

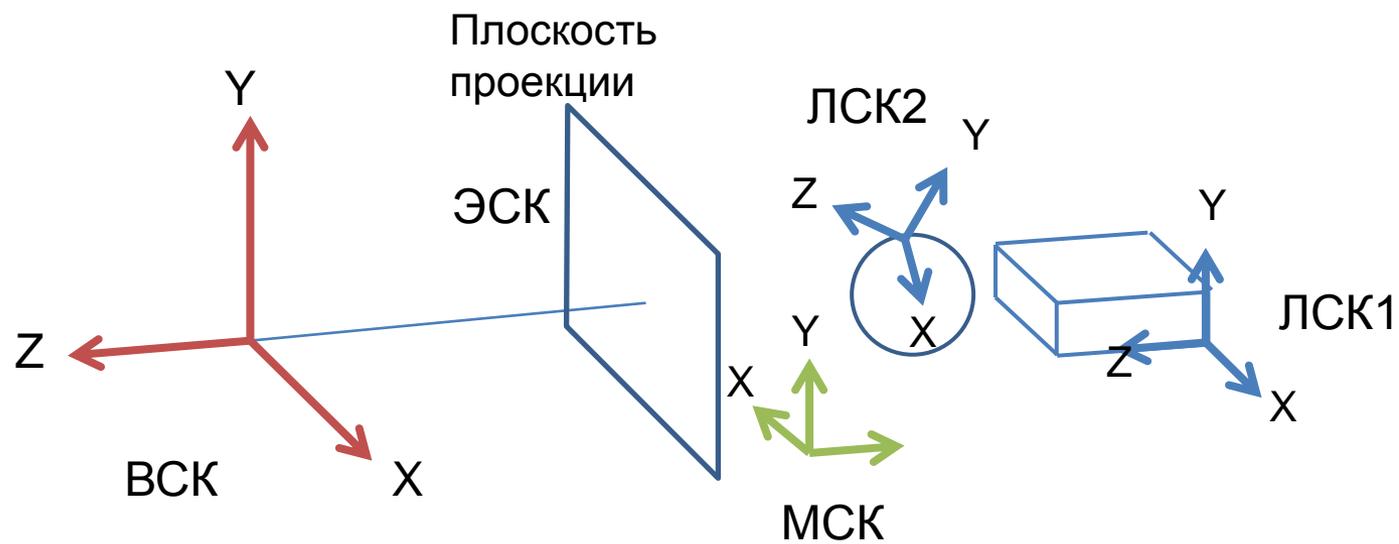
Графический процесс



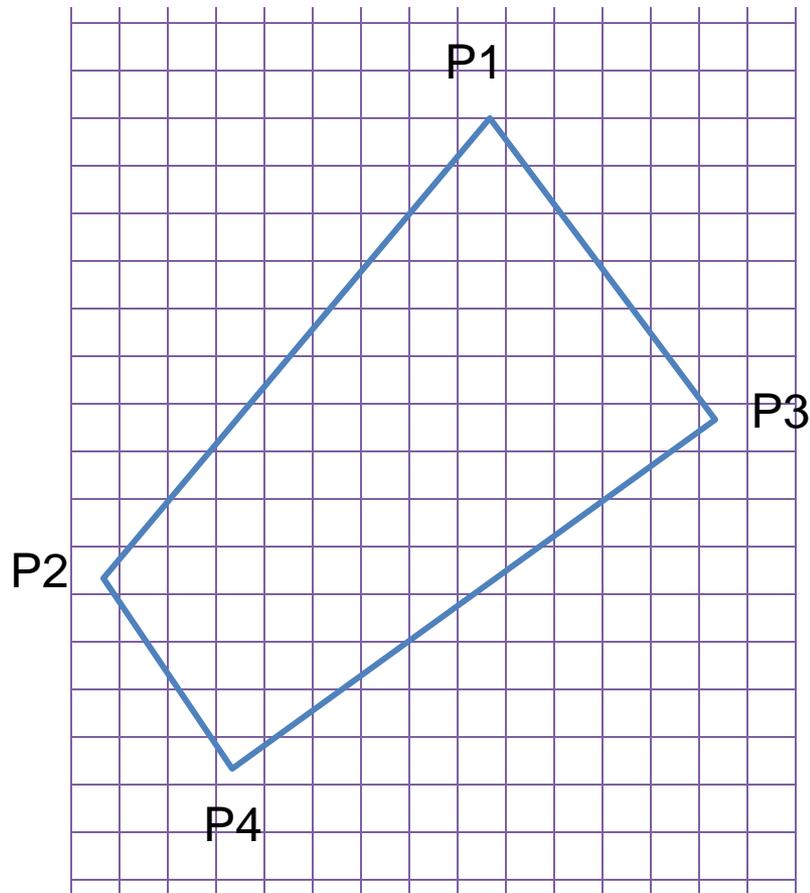
Конвейер OpenGL. Переходим к этапу операций над пикселями



Графический конвейер



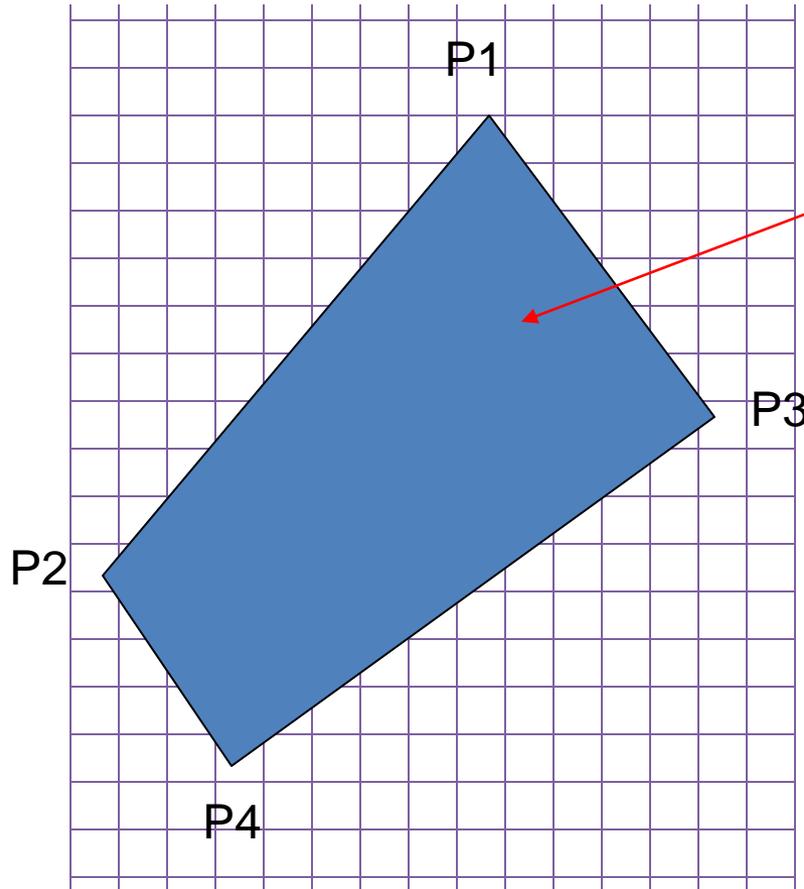
Задача этапа растеризации – рассчитать цвет пикселей, соответствующих примитиву в экранных координатах



После преобразований
получили точки
P1, P2, P3, P4

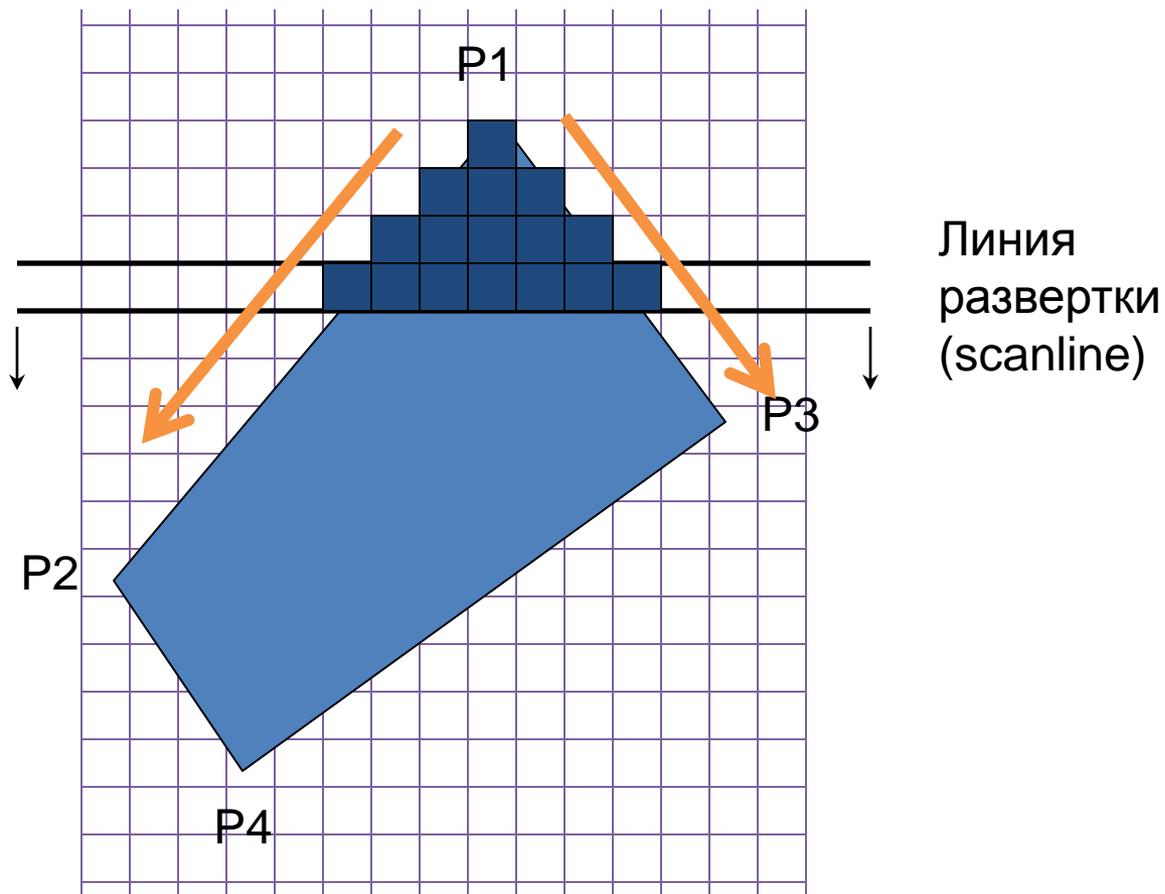
Переходим на этап
растеризации

Три основных этапа вычисления цвета

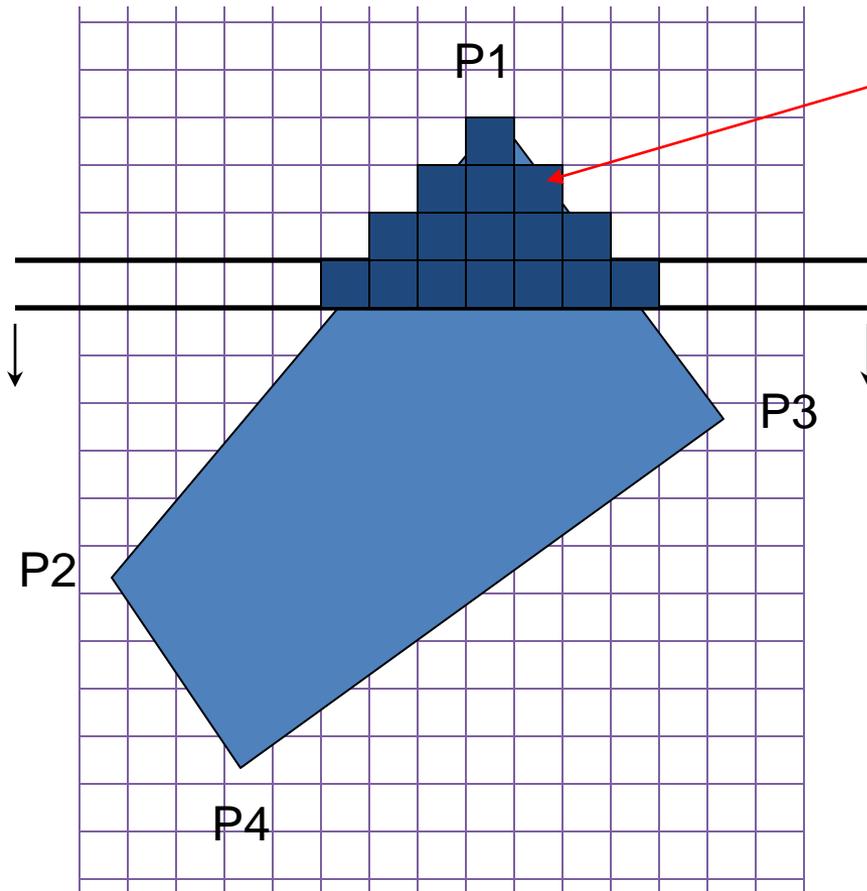


1. Растеризовать примитив
2. Вычислить цвет каждого пикселя
3. Скомбинировать с цветом фона

Растрезизация: процесс вычисления пикселей растра, принадлежащих примитиву



Вычисление цвета пикселя: материал, текстура, фон

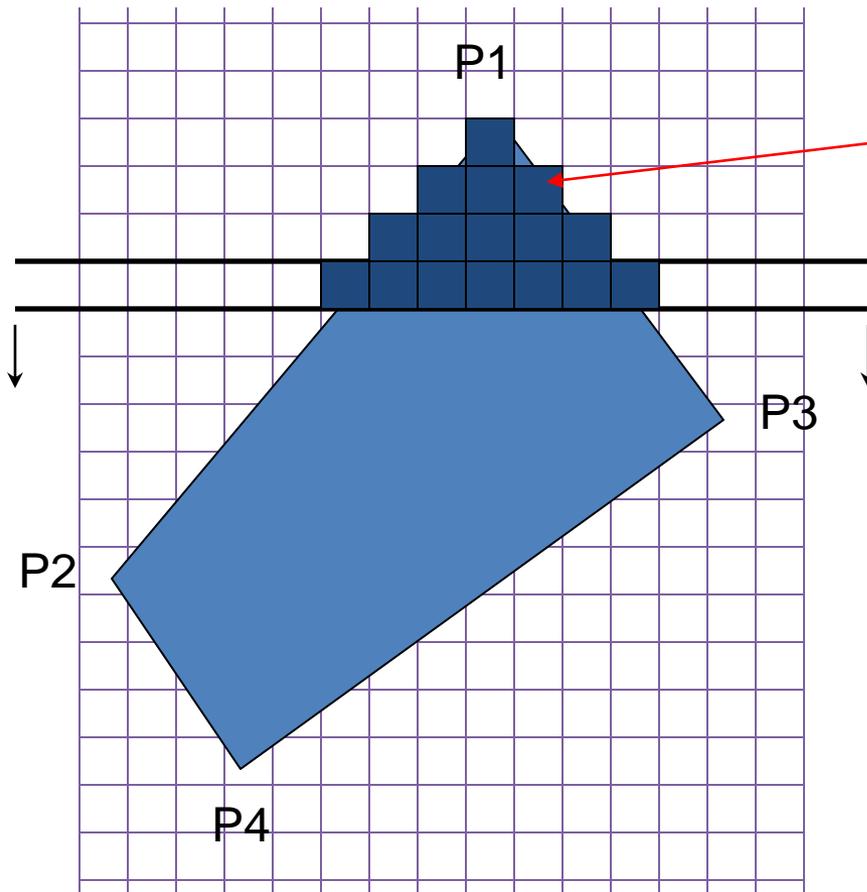


1. Цвет материала C_m
2. Цвет текстуры C_t
3. Цвет фона C_b

$$C = F1(C_f, C_b)$$

$$C_f = F2(C_m, C_t)$$

Вычисление цвета пикселя: материал посчитали, теперь нужен цвет текстуры



1. Цвет материала C_m
2. Цвет текстуры C_t
3. Цвет фона C_b

$$C = F1(C_f, C_b)$$

$$C_f = F2(C_m, C_t)$$

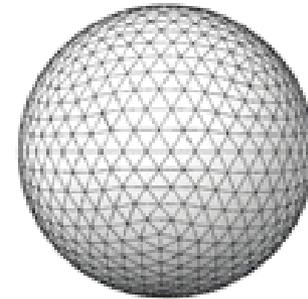
Отображение текстуры: «натягивание» изображения на 3D модель

Общее отображение текстуры

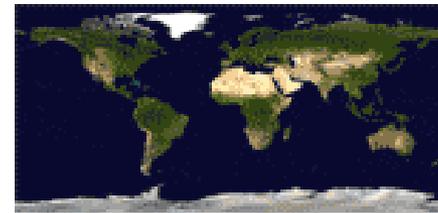
- Узор, определенный в 2D области, «наклеивается» на объект как кусок обоев и, фактически становится частью объектной базы данных.
- Когда объект перемещается, текстурный узор перемещается вместе с ним

Разнообразие технологий

- View-dependent mapping techniques.
- Bump mapping techniques.
- Displacement mapping techniques
- ...



Sphere with no texture



Texture image



Sphere with texture

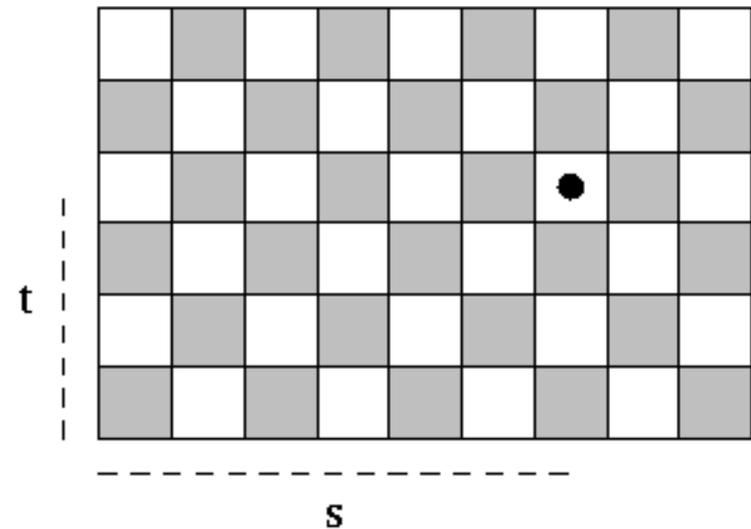
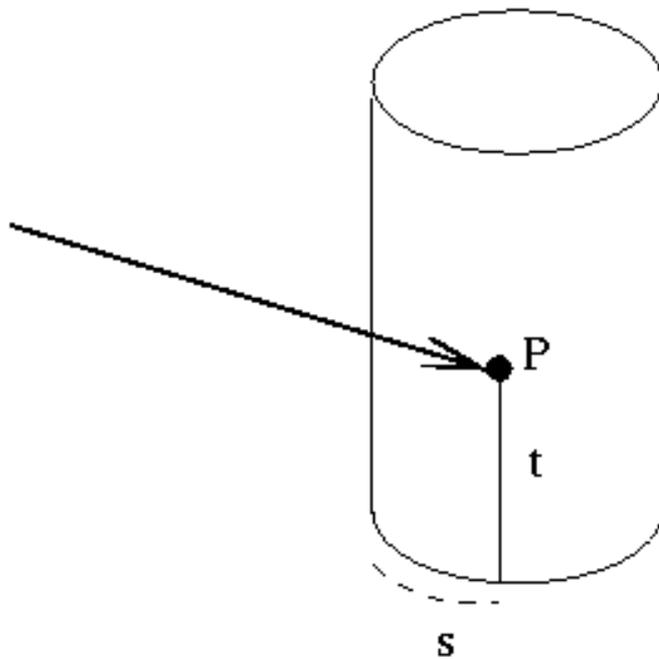
Две основные задачи: отображение и применение

Текстура – часть модели освещения и часть геометрической модели

Как осуществляется отображение 2D -> 3D?

Какой атрибут или параметр модулируется, чтобы получить желаемый эффект?

Пример отображение текстуры: обернуть изображение вокруг цилиндра



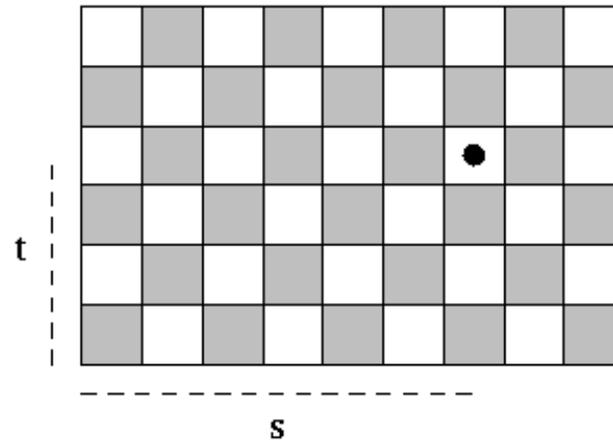
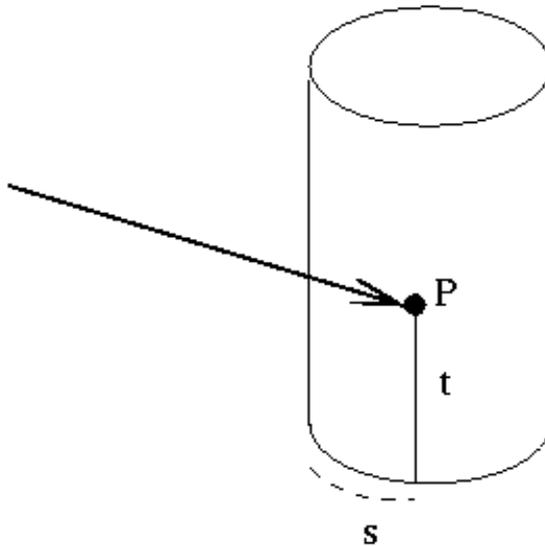
Пример отображение текстуры: считаем координаты в текстуры как (угол, высота)

P - точка нанесения текстуры

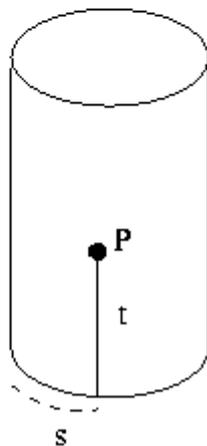
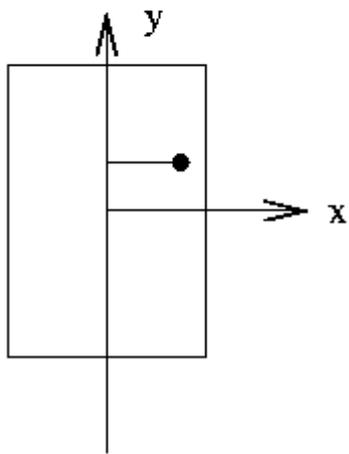
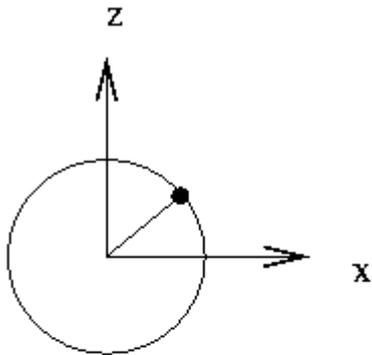
Перевести P в параметрические координаты (s,t)

s - угловая координата в радианах от оси X

t - координата высоты



Пример отображение текстуры: получение результирующих формул



Parametric Equation:

$$x = \cos 2\pi s$$

$$y = 2t - 1$$

Solve for (s, t) :

$$s = \frac{1}{2\pi} \arccos x$$

$$t = (y + 1)/2$$

Аналогично можно задать отображение текстуры на другие поверхности

Параметрические (B-сплайн) поверхности

Полигональные поверхности

Неявно заданные поверхности

Отображение текстуры для полигональных поверхностей: соответствие на вершинах

Задать соответствие между координатами изображения текстуры и точками объекта

Для полигонального представления используется соответствие на вершинах

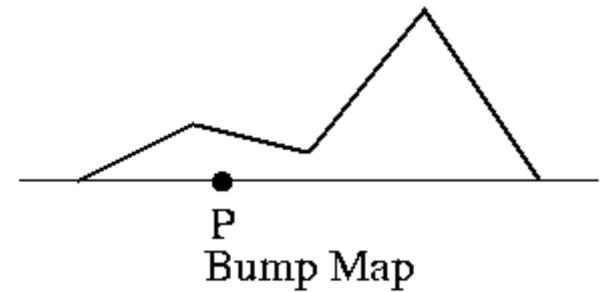
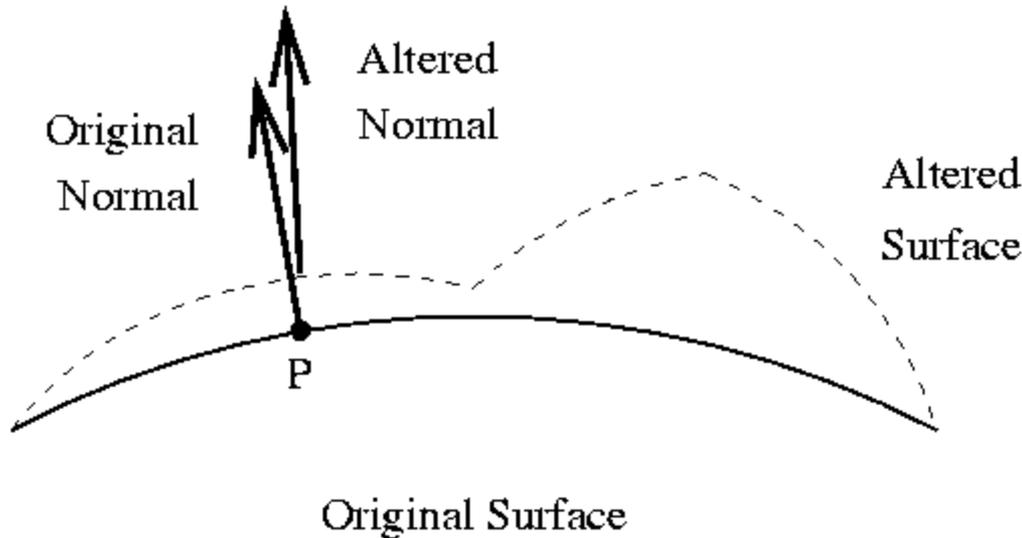
- Линейная интерполяция внутри полигона (Гуро)
- Перспективная коррекция

Цвет из текстуры может модулировать какой-либо коэффициент модели освещения

$$I_r = k_a * I_a + I_i (k_d * (N \cdot L) + k_s * (R \cdot V)^\alpha)$$

- отражение от поверхности - surface color texture
- вектор нормали N - bump mapping
- коэффициенты k_d , k_s , α - specularity mapping
- падающий свет I_i - environment mapping
- геометрия - displacement mapping
- прозрачность - transparency mapping

Bump mapping

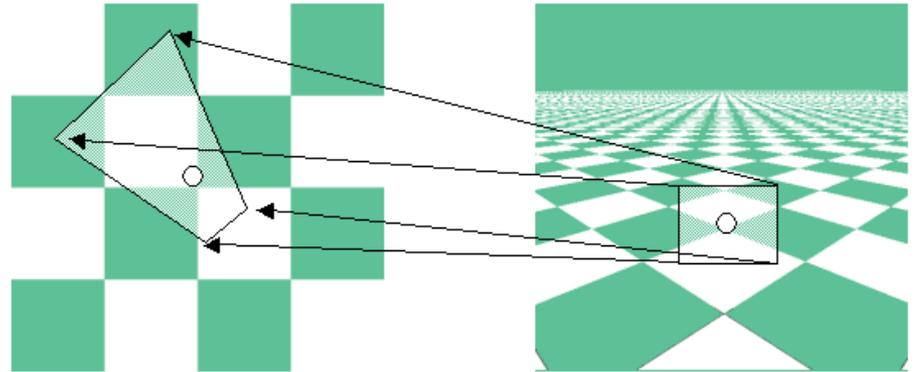
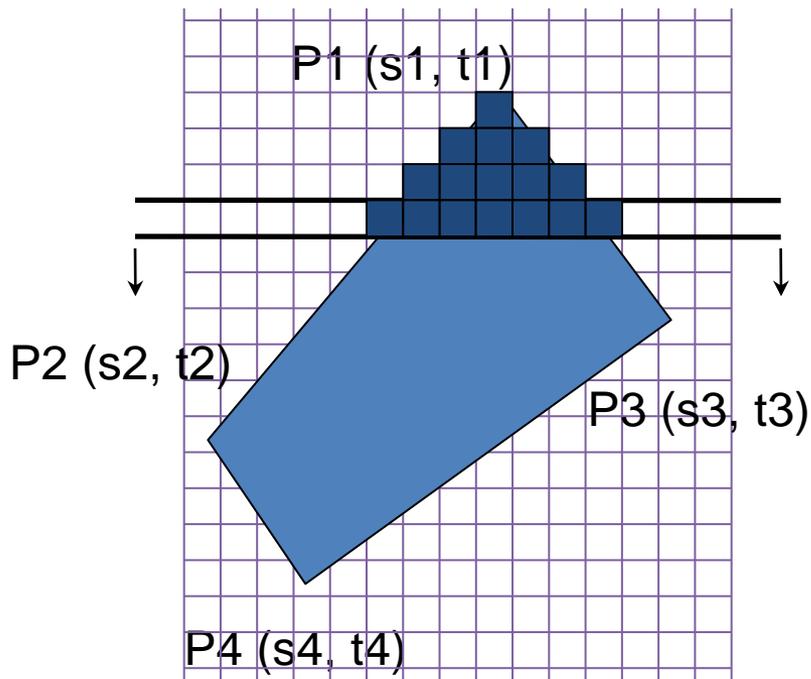


Let $B(s, t)$ be the bump map value.

We add this amount in the direction normal to $\mathbf{S}(s, t)$:

$$\mathbf{S}'(s, t) = \mathbf{S}(s, t) + B(s, t) \cdot \mathbf{n}(s, t)$$

Визуализация текстуры: для каждого пикселя находим отображение назад в текстуру

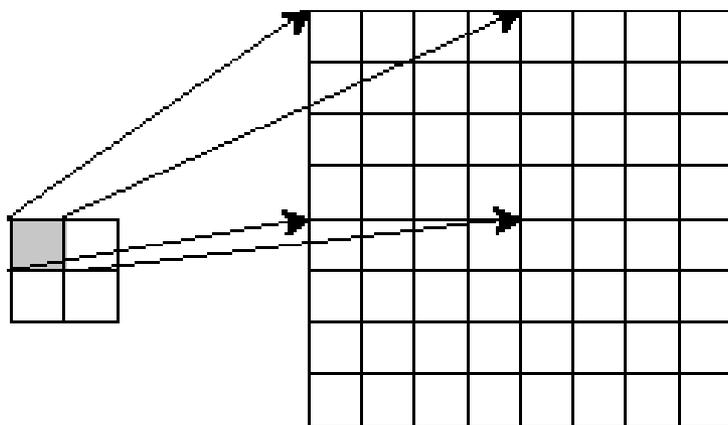


Нужна фильтрация, т.к. образ пикселя в текстуре не попадает на сетку текстуры!

Фильтрация текстур

Текстура

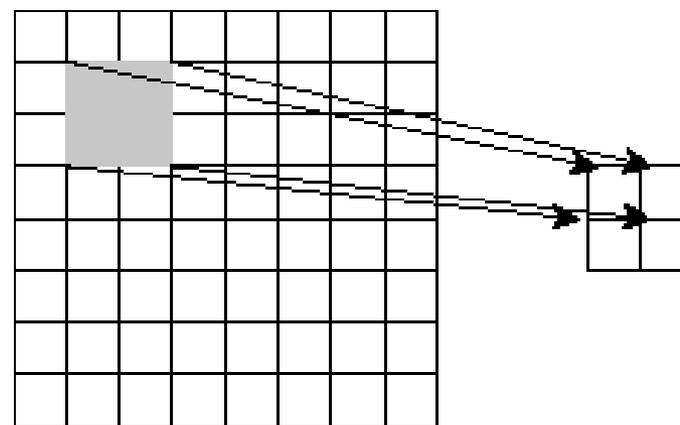
Экран



Magnification

Текстура

Экран



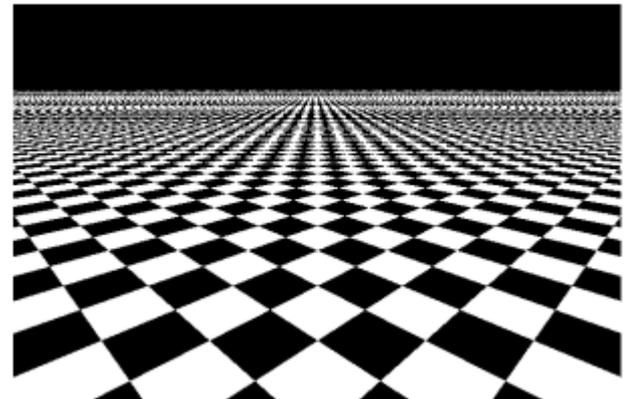
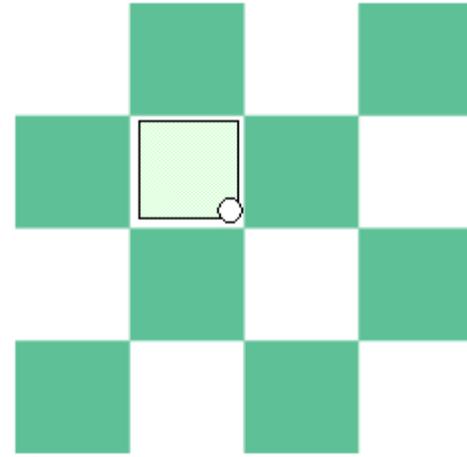
Minification

Метод ближайшего соседа: берем ближайший тексель

Выбирается цвет ближайшего соответствующего пикселя текстуры

Быстрый метод

Низкое качество



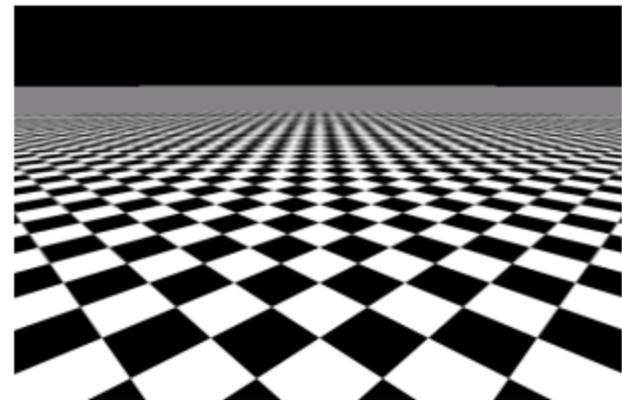
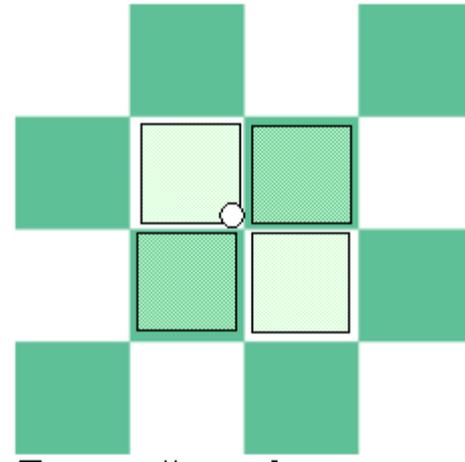
Билинейная фильтрация: усредняем ближайшие четыре пикселя

Четыре пикселя текстуры, ближайшие к текущей точке экрана

Результирующий цвет — смешение цветов этих пикселей

Быстро, достаточно качественно

За исключение случаев, когда смотрим на плоскость под углом



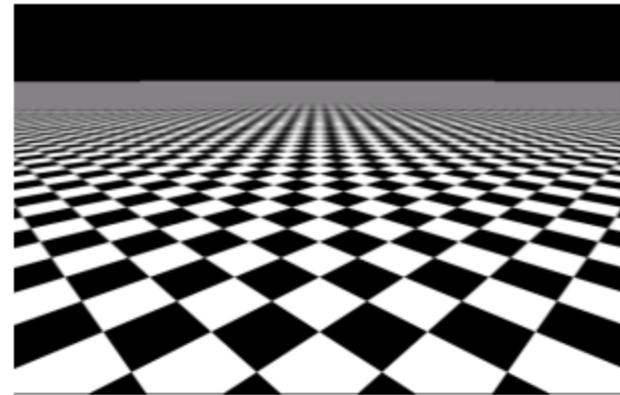
Mipmapping (multum in parvo): вычисляем разные размеры фильтра заранее

Выбирается подходящий
уровень мипмапа

- чем больше размер "образа"
пикселя в текстуре, тем меньший
мипмап берется

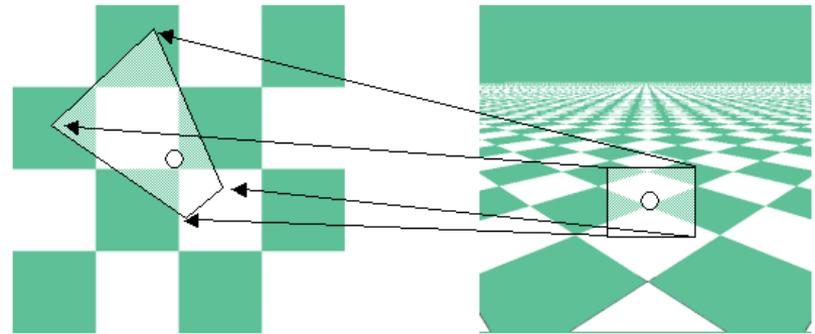
Далее значения в мипмапе
могут усредняться билинейно
или методом ближайшего
соседа

Дополнительно происходит
фильтрация между соседними
уровнями мипмапа
(трилинейная фильтрация)

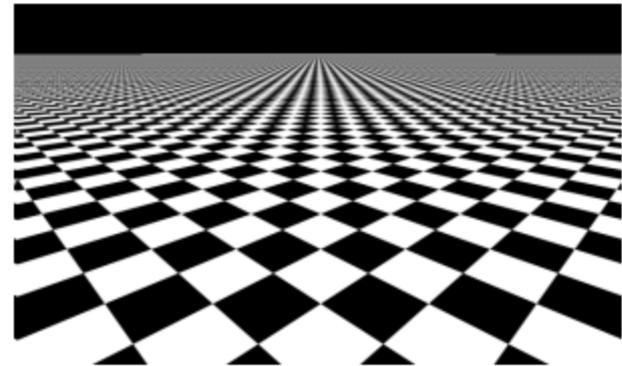


Анизотропная фильтрация: вытянутый фильтр

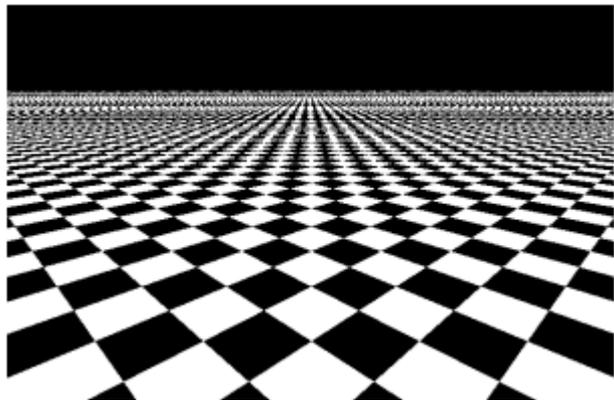
Вместо квадратного
фильтра, используется
вытянутый



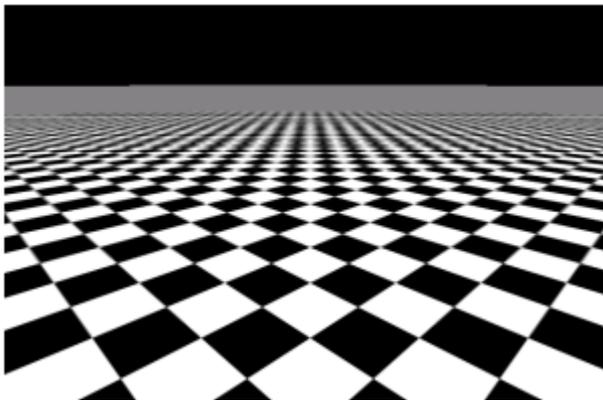
Позволяет более
качественно выбрать
нужный цвет для
экранного пикселя



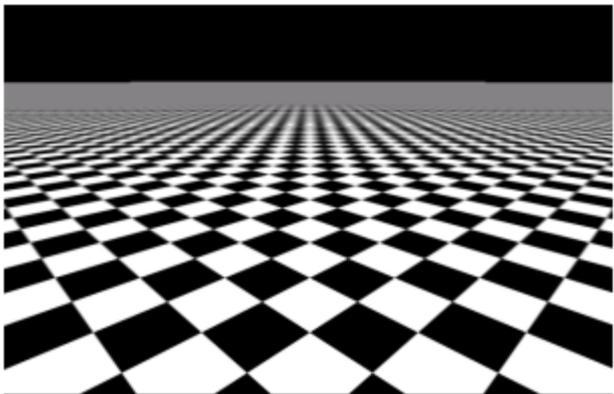
Сравнение фильтраций



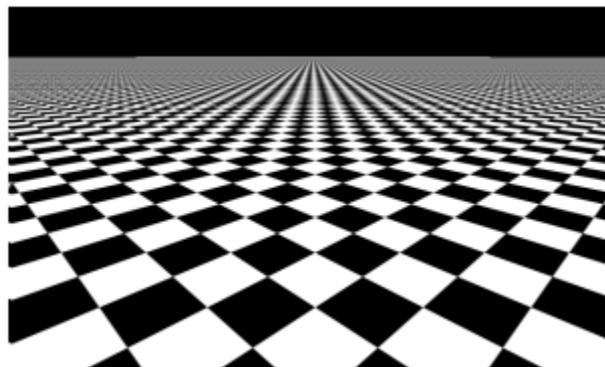
Ближайший сосед



Билинейная

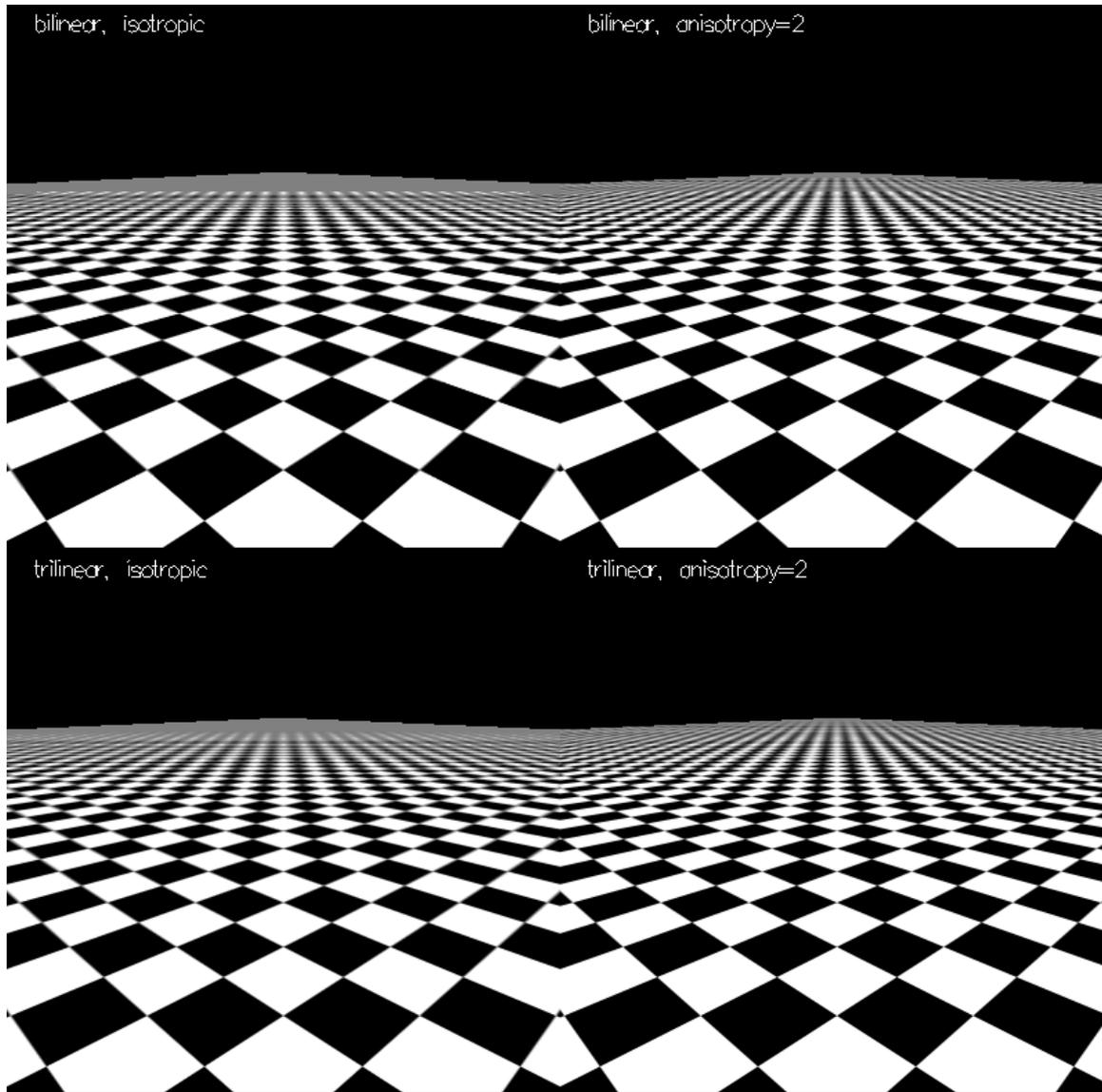


Трилинейная

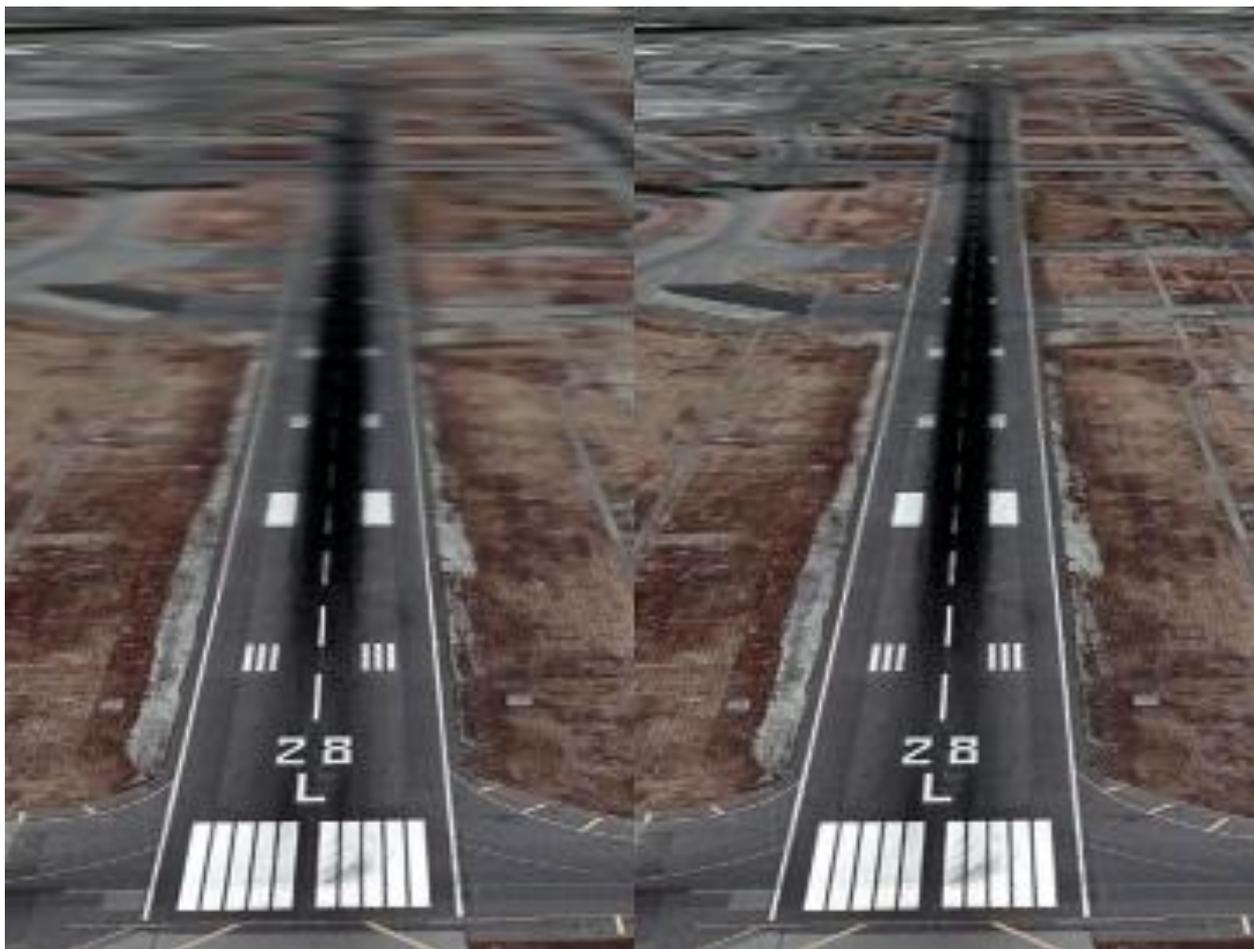


Анизотропная

Анизотропная vs. Изотропная



Еще сравнение



Trilinear

Anisotropic

Текстурирование в OpenGL

- Создаем текстуру - прямоугольный массив с цветами пикселей. Высота и ширина должны быть степенями двойки.

RGB₀₀	RGB₁₀	...	RGB_{N0}
RGB₀₁	RGB₁₁	...	RGB_{N1}
...
RGB_{0M}	RGB_{1M}	...	RGB_{NM}

$$N = 2^n - 1,$$

$$M = 2^m - 1.$$

- Получаем номер текстурного объекта

```
GLuint texture;  
glGenTextures(1, &texture);
```

- Активизируем текстурный объект

```
glBindTexture(texture);
```

Текстурирование в OpenGL (2)

□ Загружаем текстуру

```
glPixelStorei(GL_UNPACK_ALIGNMENT,1);
glTexImage2D(GL_TEXTURE_2D,
             0, // Mip-level
             GL_RGB, // Формат текстуры
             tex_width, tex_height,
             0, // Ширина границы
             GL_RGB, // Формат исходных данных
             GL_UNSIGNED_BYTE, // Тип данных
             tex_bits); // Исходные данные
```

□ Устанавливаем режимы текстурирования

```
glTexParameteri(GL_TEXTURE_2D,
                 GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D,
                 GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D,
                 GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,
                 GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

Текстурирование в OpenGL (3)

- ❑ Разрешаем текстурирования

```
glEnable(GL_TEXTURE_2D);
```

- ❑ Задаем текстурные координаты

```
glTexCoord2d(u, v);
```

- ❑ Возвращаем номер текстурного объекта в список свободных

```
glDeleteTextures(1, &texture);
```

Автоматическая генерация текстурных координат

- Автоматическая генерация текстурных координат
 - `glTexGen{ifd}[v]()`
- Режимы генерации координат:
 - `GL_OBJECT_LINEAR`
 - `GL_EYE_LINEAR`
 - `GL_SPHERE_MAP`
- Координаты считаются как расстояние от заданной плоскости $Ax+By+Cz+D=0$

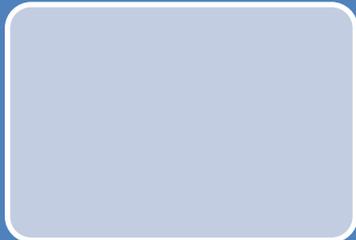
Три основных темы, все связаны с этапом растеризации



Текстурирование



Удаление невидимых поверхностей

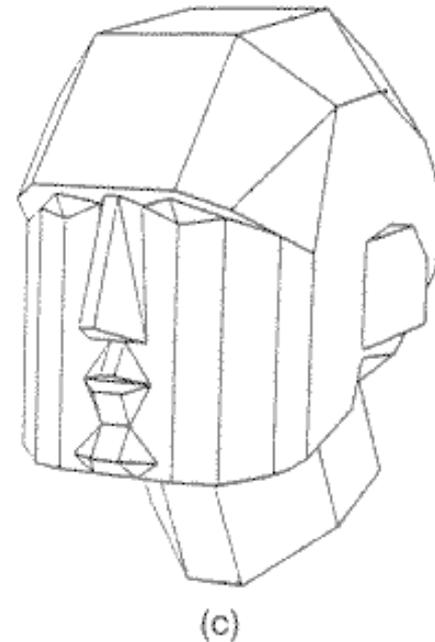
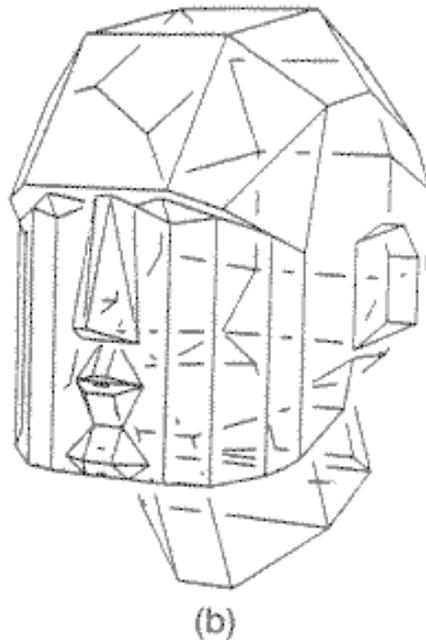
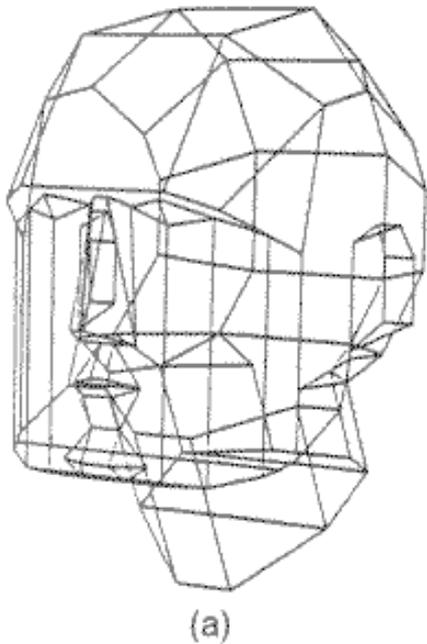


Композиты

Удаление невидимых линий и поверхностей – зачем?

При проецировании на экран часто оказывается, что отдельные части объектов (или объекты полностью) перекрываются другими объектами и не видны наблюдателю

Метод растеризации (преобразования координат) сам по себе не позволяет ответить на вопрос, какие объекты видны



Классификация методов

По способу изображения объекта

- Каркасное
- Сплошное

По пространству, в котором решается задача

- Мировое пространство
- Видовое пространство
- Картинная плоскость

По точности решения

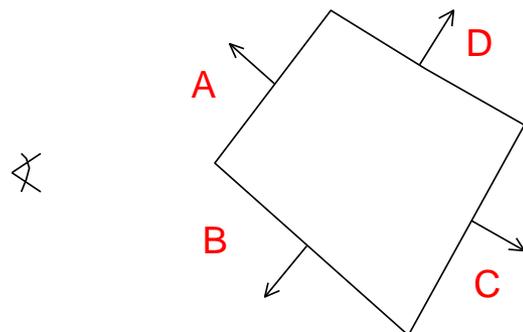
- Аналитическое
- Приближенное

Методы удаления невидимых поверхностей

- Удаление нелицевых граней
- Трассировка лучей
- Алгоритм художника
- Буфер глубины

Лицевые и нелицевые грани

Если исходная геометрия описывает **сплошные тела**, то для каждой грани можно определить вектор внешней нормали



Нормали к граням A и B смотрят в сторону наблюдателя

- наблюдатель находится в положительном полупространстве по отношению к плоскости, проходящей через соответствующую грань

Такие грани называются **лицевыми** (front-faced). Грани C,D - **нелицевые** (back-faced)

Свойства (не-)лицевых граней

I. Для сплошных (solid) тел, ни одна из нелицевых граней никогда не будет видна даже частично

- При определении видимости нелицевые грани можно всегда отбрасывать
- сокращает число граней для визуализации примерно вдвое

II. Если вся сцена состоит только из одного выпуклого объекта, все лицевые грани и только они будут видны, причем полностью

Удаление нелицевых граней: простой, но только для выпуклых объектов

Метод удаления нелицевых граней:

- (+) очень простой, может сократить число граней для растеризации в два раза
- (-) работает только для сплошных тел
- (-) удаляет все невидимые грани только для сцен с одним выпуклым сплошным объектом, иначе должен использоваться в сочетании с другими методами

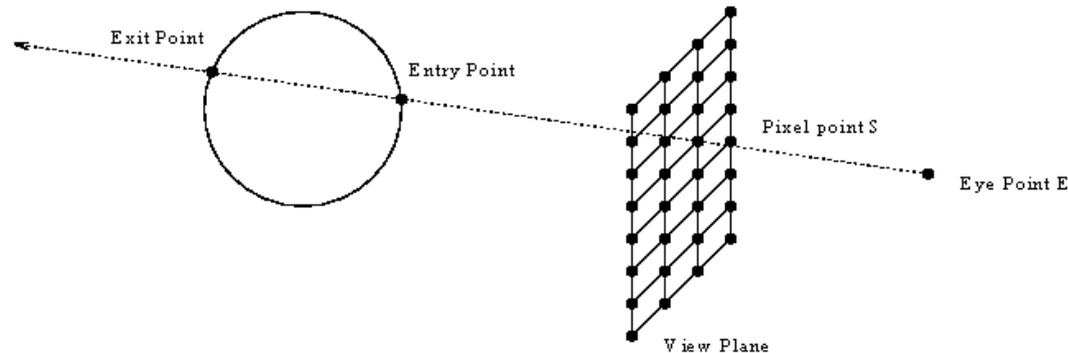
Трассировка лучей: неявно решает задачу удаления невидимых граней

При использовании метода трассировки лучей через каждый пиксель картинной плоскости выпускается луч (из положения наблюдателя) в сцену.

Далее находится ближайшее пересечение этого луча с объектами сцены

Оно определяет, что именно будет видно в данном пикселе

A Simple Example: Sphere with Diffuse Reflection



Трассировка лучей: работает хорошо, но не подходит для растеризации

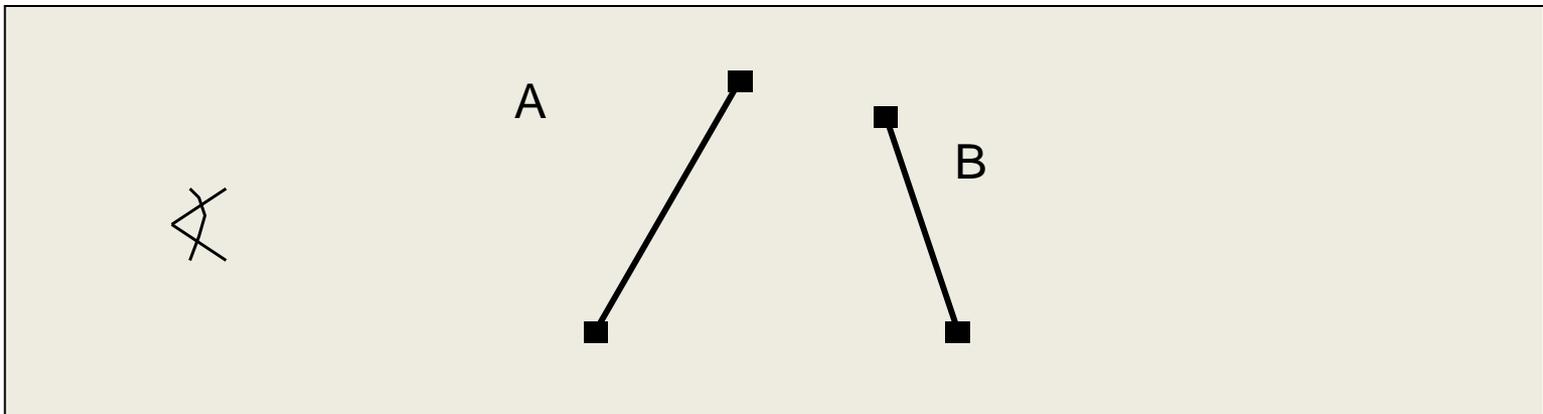
Метод трассировки лучей

- (+) работает для любой геометрии
- (-) только для визуализации методом трассировки лучей, т.е. не подходит для растеризации

Алгоритм художника: рисуем грани от дальних к ближним

Алгоритм художника (painter's algorithm)

- Явно сортирует все грани сцены в порядке их приближения к наблюдателю (back-to-front)
- Выводит грани в этом порядке
- => получается корректное изображение

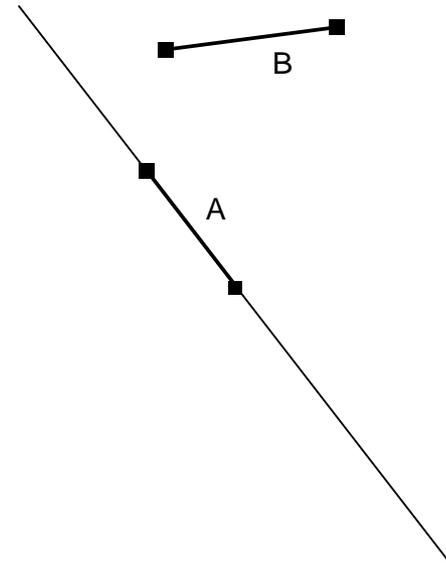


Алгоритм художника по шагам

1. Упорядочить все многоугольники в соответствии с наименьшей (дальней) координатой z
2. Разрешить неоднозначности, которые вызывают наложение z -габаритов
3. Нарисовать многоугольники в порядке возрастания наименьшей координаты z

Упорядочивание граней: полупространства для определения порядка граней

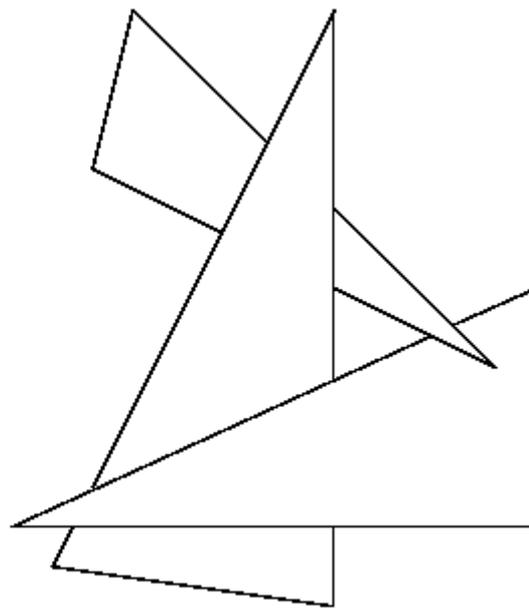
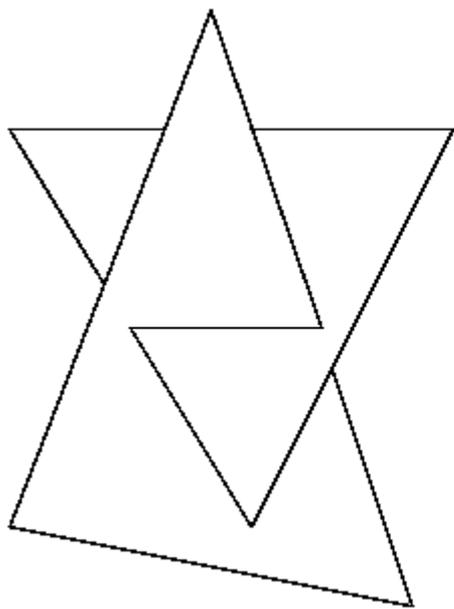
- Грань В не может закрывать грань А от наблюдателя
- Т.к находится в другом полупространстве относительно плоскости, проходящей через грань А.



Пять тестов алгоритма художника

1. Накладываются ли x -габариты m -ков?
2. Накладываются ли y -габариты m -ков?
3. P полностью за плоскостью Q по отношению к наблюдателю?
4. Q полностью перед плоскостью P по отношению к наблюдателю?
5. Пересекаются ли проекции многоугольников на плоскость (x, y) ?

Проблемы алгоритма художника: не всегда можно однозначно упорядочить



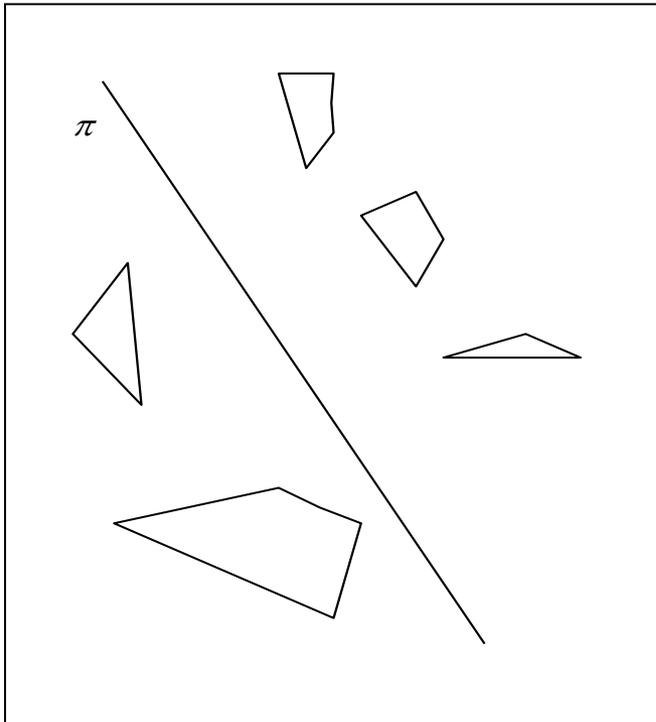
Достоинства и недостатки

Алгоритм художника:

- (-) линейная сложность сортировки
- (-) квадратичная сложность для многоугольников с пересекающимися z-объемами
- (-) не может справиться со всеми типами взаимного расположения

Метод двоичного разбиения пространства: та же идея, что в алгоритме художника

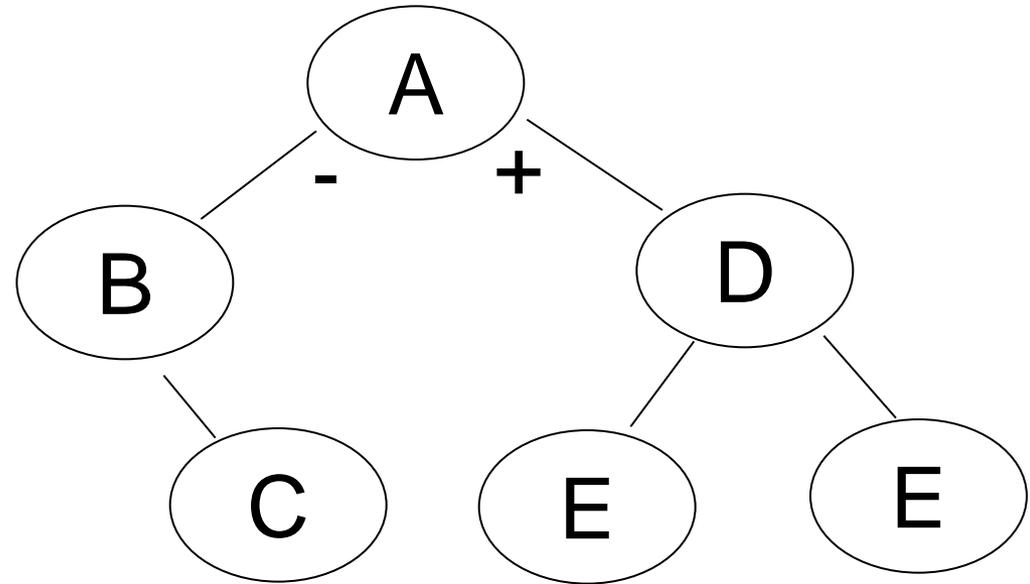
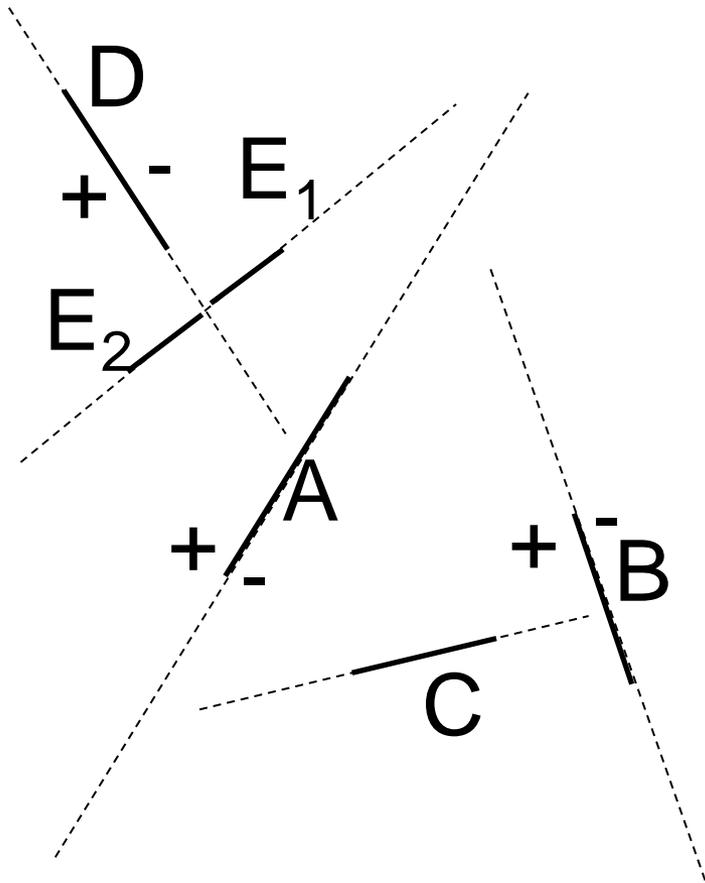
Пусть известно, что плоскость π разбивает все грани (объекты) сцены на два непересекающихся множества в зависимости от того, в каком полупространстве по отношению к данной плоскости они лежат



Тогда ни одна из граней, лежащих в том же полупространстве, что и наблюдатель, не может быть закрыта ни одной из граней из другого полупространства

=> удалось осуществить частичное упорядочение граней исходя из возможности загораживания друг друга.

Метод двоичного разбиения пространства: построение дерева



Метод двоичного разбиения пространства : алгоритм определения последовательности

```
class BSPNode {  
    Face *face; // Грань объекта  
    BSPNode *positive;  
    BSPNode *negative;  
    ...  
}
```

```
void BSPNode::Draw() {  
    if(face->Sign(viewer) == 1) {  
        if(negative) negative->Draw();  
        face->Draw();  
        if(positive) positive->Draw();  
    } else {  
        if(positive) positive->Draw();  
        face->Draw();  
        if(negative) negative->Draw();  
    }  
}
```

Достоинства и недостатки

Метод двоичного разбиения пространства

- (+) логарифмическая сложность сортировки объектов по глубине (быстро работает), позволяет быстро выводить полигоны back-to-front
- (-) проблемы, аналогичные алгоритму художника
- (-) работает для статических сцен, дерево перестраивается долго

Буфер глубины

Каждому пикселю картинной плоскости, кроме значения цвета, хранящемуся в буфере кадра, сопоставляется еще значение глубины

- расстояние вдоль направления проецирования от картинной плоскости до соответствующей точки пространства

```
foreach(p in pixels)
  if(p.z < zBuffer[p.x, p.y])
  {
    draw( p );
    zBuffer[p.x, p.y] = p.z;
  }
```

Примечание: для работы буфера глубины нужно проективное преобразование

Если бы использовались вырожденные матрицы проекции вместо проективного преобразования, вычислить значение глубины было бы невозможно!

Проективное преобразование преобразует видимую часть сцены в стандартную область видимости, т.е. $z \rightarrow [-1, 1]$

Используется как значения в z-буфере!

Достоинства и недостатки

Метод буфера глубины

- (+) очень простой, легко реализуется аппаратно
- (+) работает для произвольной геометрии
 - необязательно с граничным представлением
- (-) требует дополнительную память
- (-) требует растеризации всех примитивов
- (-) не сортирует back-to-front, невозможно реализовать прозрачность

Поддержка методов удаления невидимых поверхностей в OpenGL

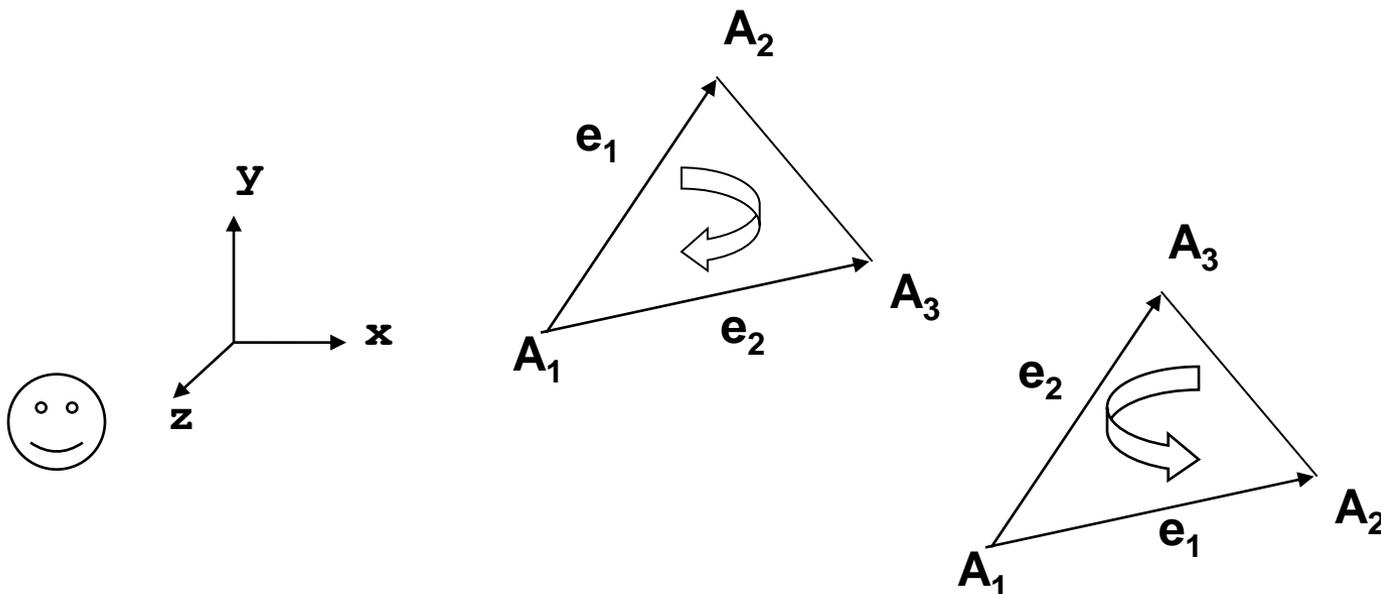
OpenGL напрямую поддерживает два метода удаления невидимых поверхностей

- удаление нелицевых граней
- буфер глубины (z-буфер)

Это не значит, что другие методы нельзя использовать!

Их нужно реализовывать самостоятельно

Лицевые и нелицевые грани



$$e_1 = A_2 - A_1,$$
$$e_2 = A_3 - A_1,$$
$$n = [e_1, e_2]$$

```
glEnable(GL_CULL_FACE); glDisable(GL_CULL_FACE);
```

```
void glFrontFace(GLenum type);  
    type = {GL_CW|GL_CCW}
```

```
void glCullFace(GLenum type);  
    type = {GL_FRONT|GL_BACK (по умолчанию)}
```

Z-буфер

- ❑ Необходимо создать z-буфер

```
glutDisplayMode (GLUT_DEPTH | /*...*/);
```

- ❑ Перед рисованием сцены очистить z-буфер

```
glClear (GL_DEPTH_BUFFER_BIT | /*...*/);
```

- ❑ Включить или выключить сравнение z координат

```
glEnable (GL_DEPTH_TEST);  
glDisable (GL_DEPTH_TEST);
```

- ❑ Возможно включить или выключить запись в z-буфер

```
glDepthMask (TRUE); или glDepthMask (FALSE);
```

- ❑ Возможно задать операцию сравнения

```
glDepthFunc (GLenum type);  
type = {GL_ALWAYS | GL_NEVER | GL_LESS | GL_GREATER |  
        GL_EQUAL | GL_NOTEQUAL | GL_LEQUAL | GL_GEQUAL}
```

Три основных темы, все связаны с этапом растеризации



Текстурирование



Удаление невидимых поверхностей



Композиты

Композирование

- **Композирование**

- Объединение двух или более независимо подготовленных изображений в одно изображение, обычно с помощью попиксельного вычисления полупрозрачности (альфа-канал), иногда с учетом глубины (z-координаты)

- Смешение, Монтаж (Compositing)

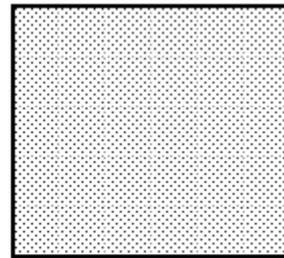
Альфа-канал кодирует информацию о частичном покрытии пикселей

- Кодирует информацию о покрытии пикселя:
 - $\alpha = 0$: нет покрытия = прозрачный
 - $\alpha = 1$: полное покрытие = непрозрачный
 - $0 < \alpha < 1$: частичное покрытие = полупрозрачный
- Пример: $\alpha = 0.3$



Partial
Coverage

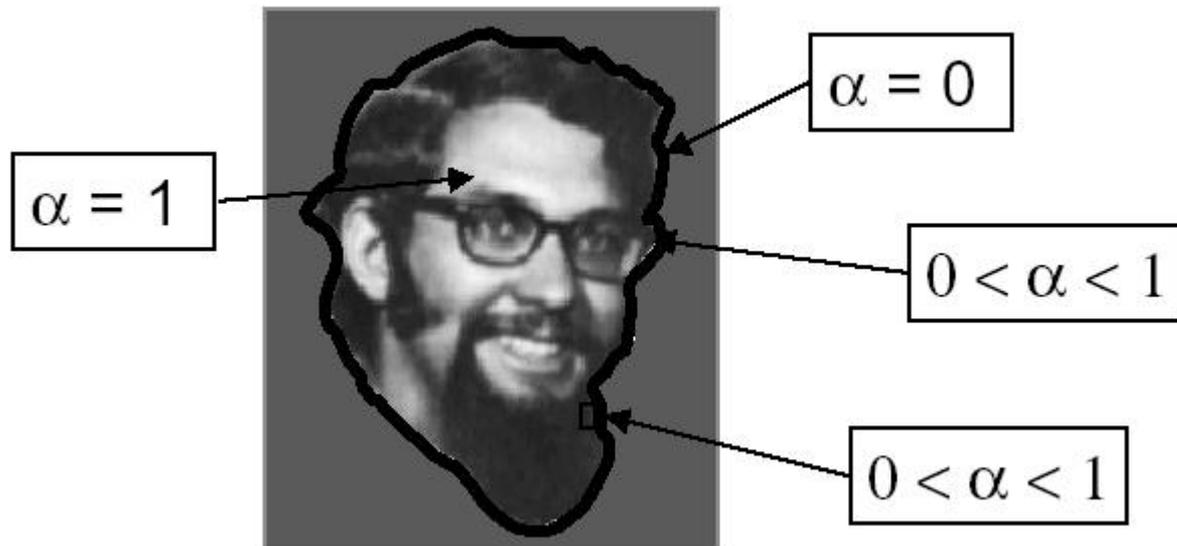
or



Semi-
Transparent

Композирование с альфа-каналом: линейная интерполяция цветов пикселей фона и изображения

Альфа-канал управляет линейной интерполяцией фоновых и передних пикселей при композировании



Полупрозрачные объекты: комбинировать просто

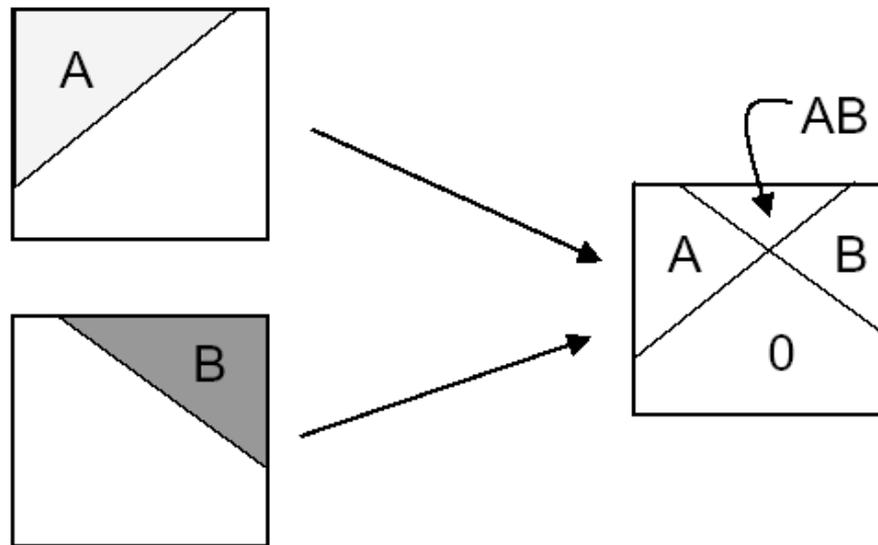
Пусть мы рисуем изображение А поверх В поверх фона G

Насколько В видно через А?	$1-\alpha_A$
Насколько А блокирует В ?	α_A
Насколько G видно через А и В ?	$(1-\alpha_A) (1-\alpha_b)$

Непрозрачные объекты: комбинировать сложнее, надо учитывать частичное покрытие пикселей

Как мы можем комбинировать два частично перекрытых пикселя?

- 3 возможных цвета (0, A, B)
- 4 региона (0, A, B, AB)



Композитная алгебра Портера-Даффа

12 возможных комбинаций



clear



A



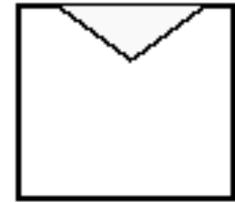
B



A over B



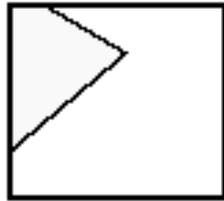
B over A



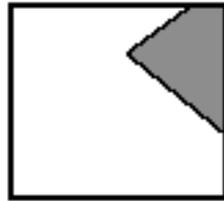
A in B



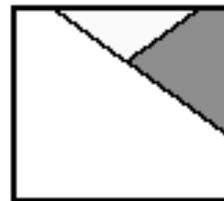
B in A



A out B



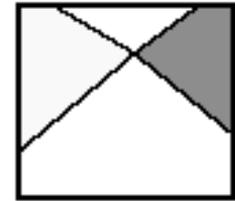
B out A



A atop B

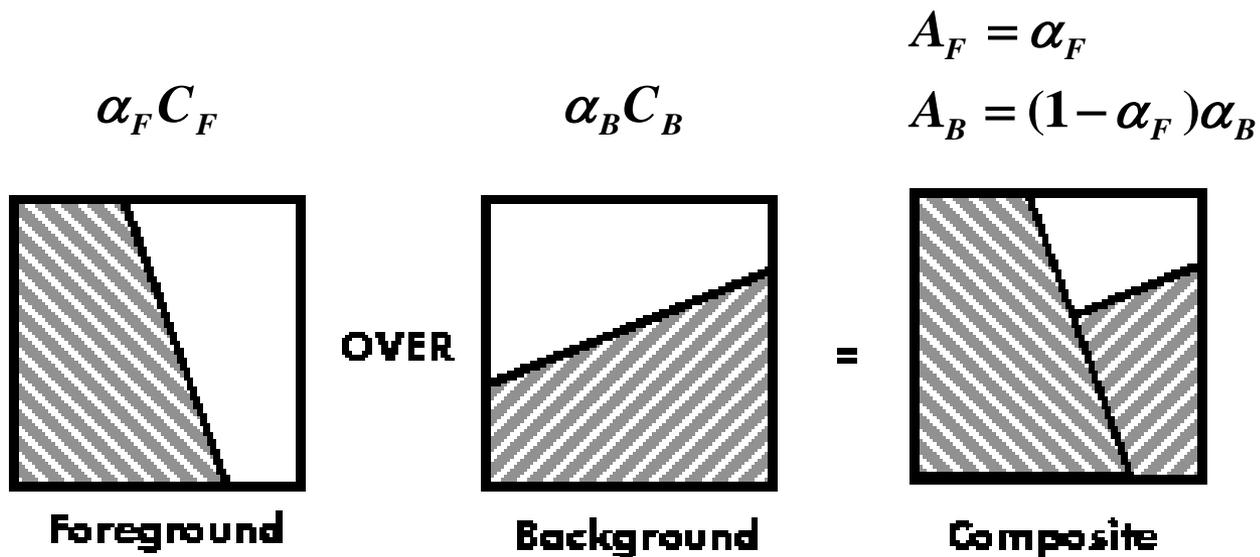


B atop A



A xor B

Оператор OVER: смешивание двух частично покрытых пикселей



$$C' = A_F C_F + A_B C_B = \alpha_F C_F + (1 - \alpha_F) \alpha_B C_B$$

$$\alpha = A_F + A_B = \alpha_F + (1 - \alpha_F) \alpha_B$$

**Оператор Over при непрозрачном фоне:
очень часто применяется при визуализации**

$$\alpha_B = 1$$

$$A_F = \alpha_F$$

$$A_B = (1 - \alpha_F)$$

\Rightarrow

$$C' = A_F C_F + A_B C_B = \alpha_F C_F + (1 - \alpha_F) C_B$$

$$\alpha = A_F + A_B = \alpha_F + (1 - \alpha_F) \alpha_B = \alpha_F + 1 - \alpha_F = 1$$

$$C' = \alpha_F C_F + (1 - \alpha_F) C_B$$

Композирование в OpenGL

Включить режим композирования

- `glEnable(GL_BLEND)`

Задать функции источника и приемника

- `glBlendFunc(source_function, dest_function)`
- Пример: оператор A over B (непрозрачный фон)
- `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`

Задать альфа-канал в вершинах или в текстуре

- `glColor(R,G,B,A);`

Выключить режим композирования

- `glDisable(GL_BLEND);`

Итоги

Удаление невидимых поверхностей

- Удаление нелицевых граней (есть в OpenGL)
- Трассировка лучей
- Алгоритм художника
- Двоичное разбиение пространства
- Буфер глубины (есть в OpenGL)

Текстурирование

- Отображение изображения на поверхность модели
- Различные виды текстурирования

Композирование

- Способ комбинации пикселей растеризуемого примитива и фона
- Композитная алгебра Портера-Даффа