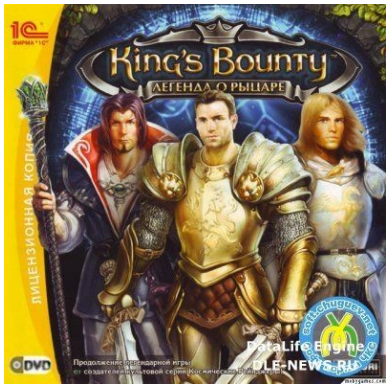
3 измерения на 3 пальцах

Андрей Аксенов, экс-повелитель шейдеров
Заявление Разработчик Дни 2010, Ярославль, экс-СССР

Призыв 1999 – ДМБ 2000

Призыв 2003 – ДМБ 2008



Application Developer Days

3 пальца – обзор для начинающих

Application Developer Days

.plan

1) Треугольнички
2) Шейдеры
3) Свет

4) Тени
5) Кукольный театр
   – не влезли ;(придется в кулуарах

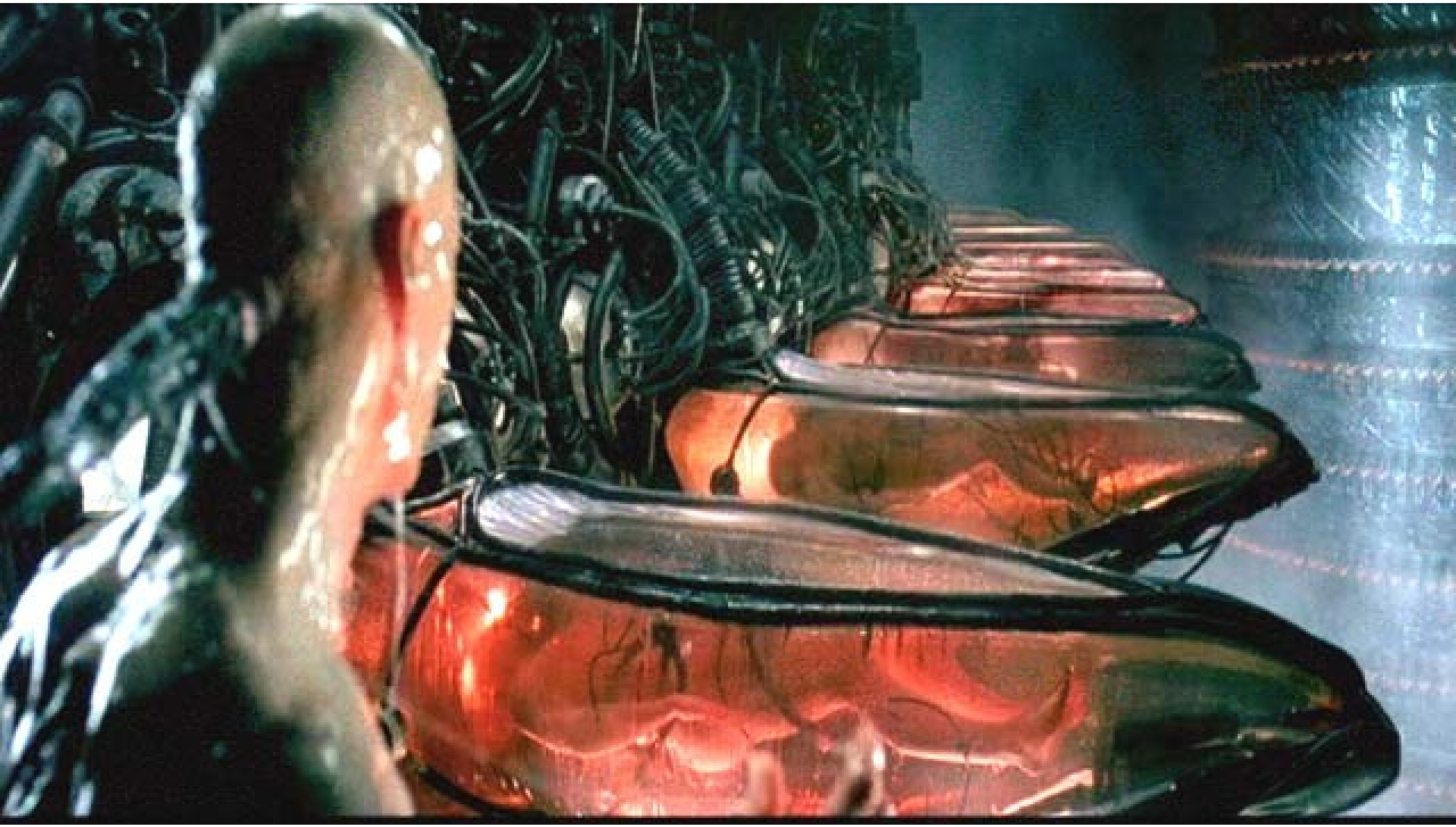

Application Developer Days


IGN.COM

3 измерения на 3 пальцах // Андрей Аксенов // ADD 2010

Application Developer Days


Application Developer Days



3 измерения на 3 пальцах // Андрей Аксенов // ADD 2010

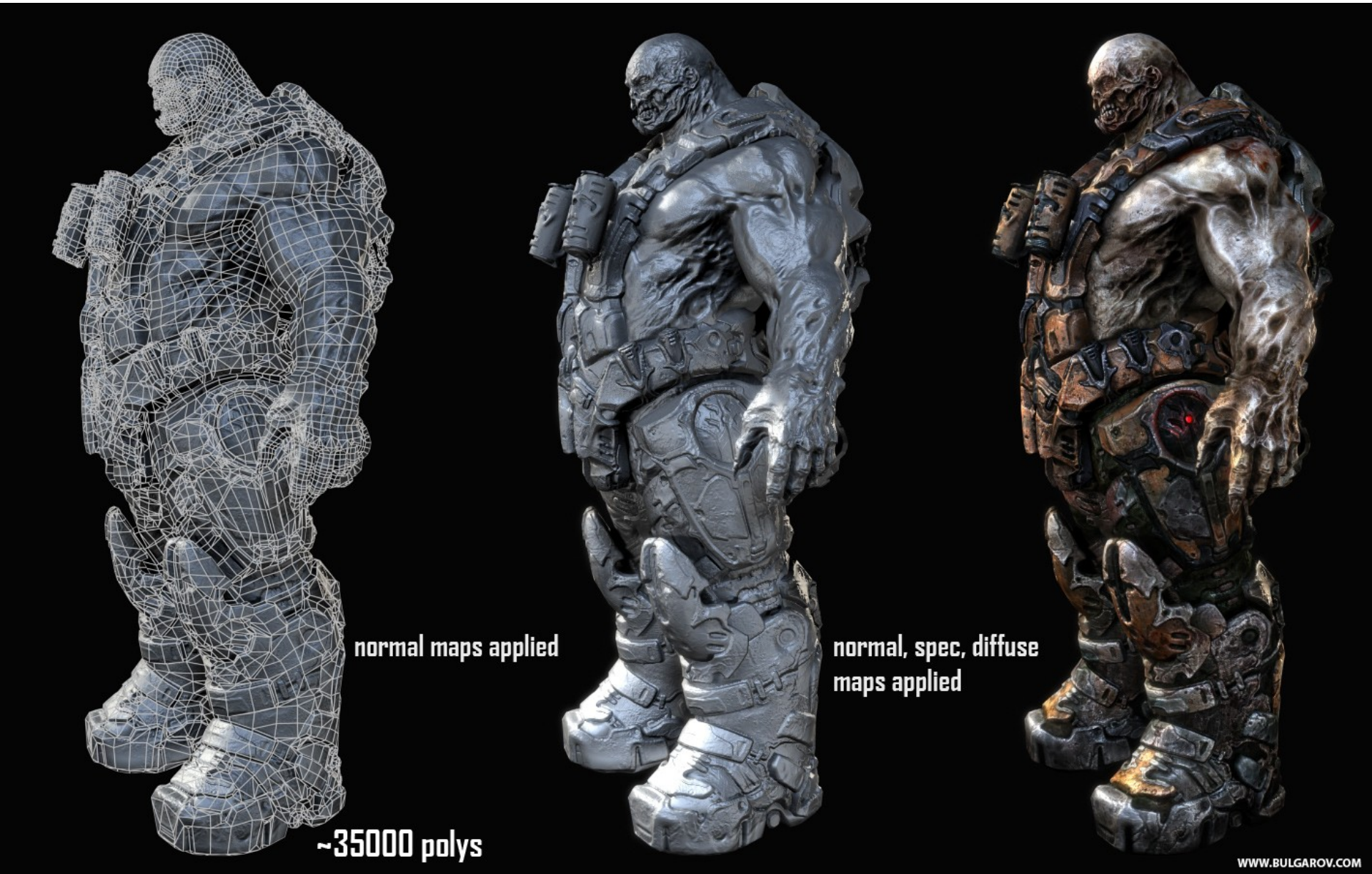
Application Developer Days

Как это устроено внутри?

- Треугольнички
- Треугольнички
- И еще треугольнички

3 измерения на 3 пальцах // Андрей Аксенов // ADD 2010

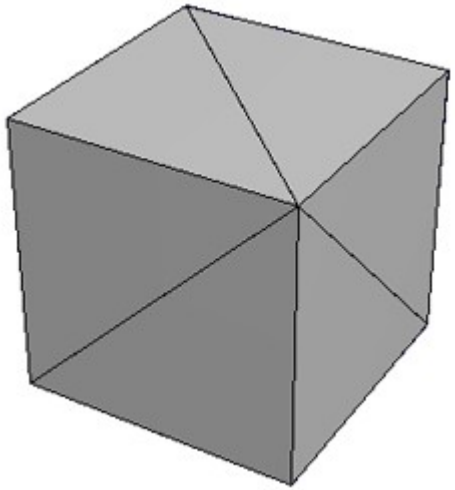
Application Developer Days



3D конвейер, 21 век

- Полная победа треугольных сеток!
  - Игры, железо (GPU)
  - CAD, CAM
  - Кино
- Легко работать, легко параллелить
- Можно сделать все, что угодно
  - Если использовать достаточно полигонов
  - REYES (Renders Everything You Ever Saw)

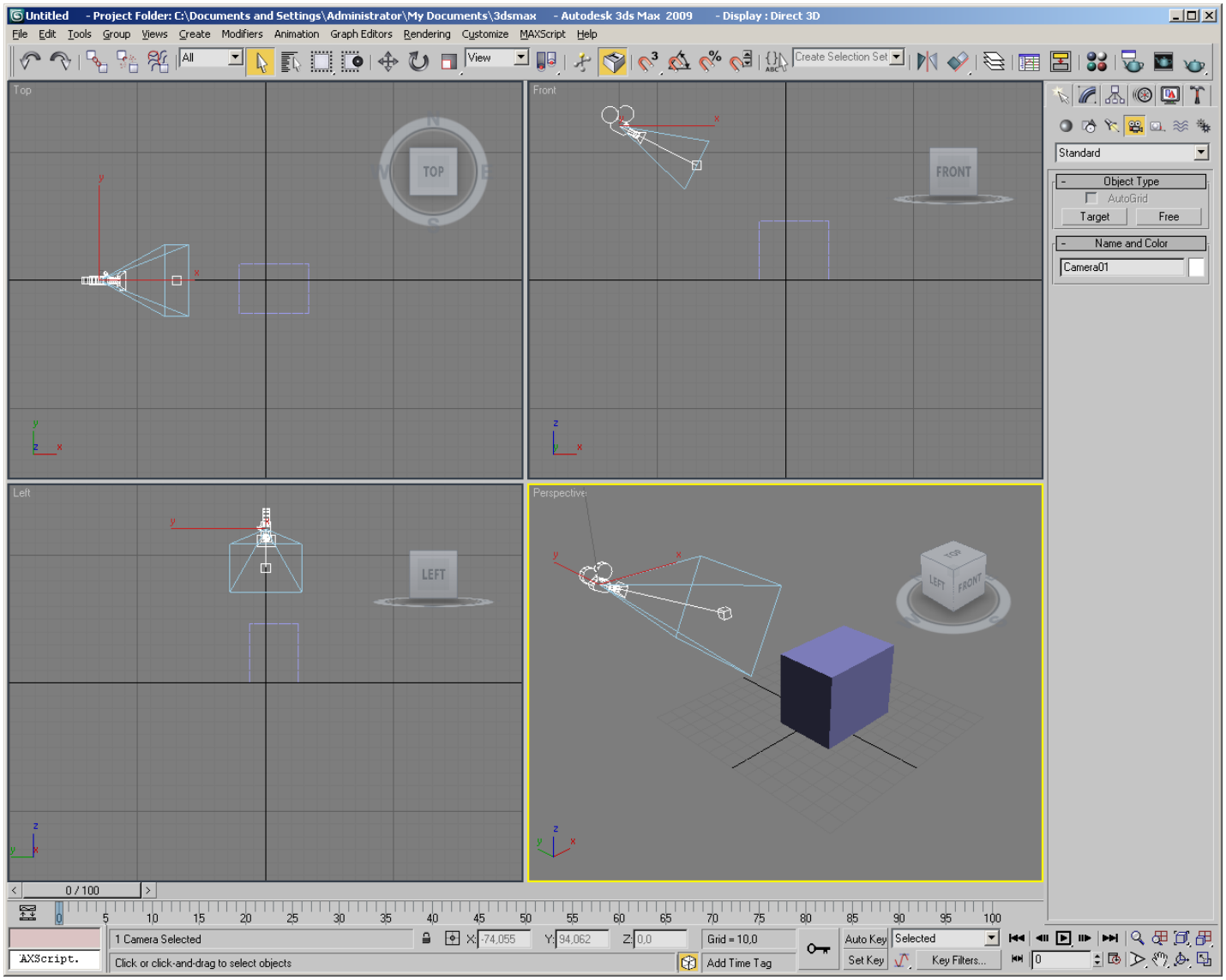
Рисуем кубик!!!



Чтобы купить что-то ненужное...

- Нужно задать объект (кубик)
  - 8 точек
  - 12 граней (6 квадратов, 12 треугольников)
- Нужно задать камеру
  - Интуитивно?
  - Где находится
  - Куда смотрит
  - FOV, Aspect Ratio, Roll

3 измерения на 3 пальцах // Андрей Аксенов // ADD 2010



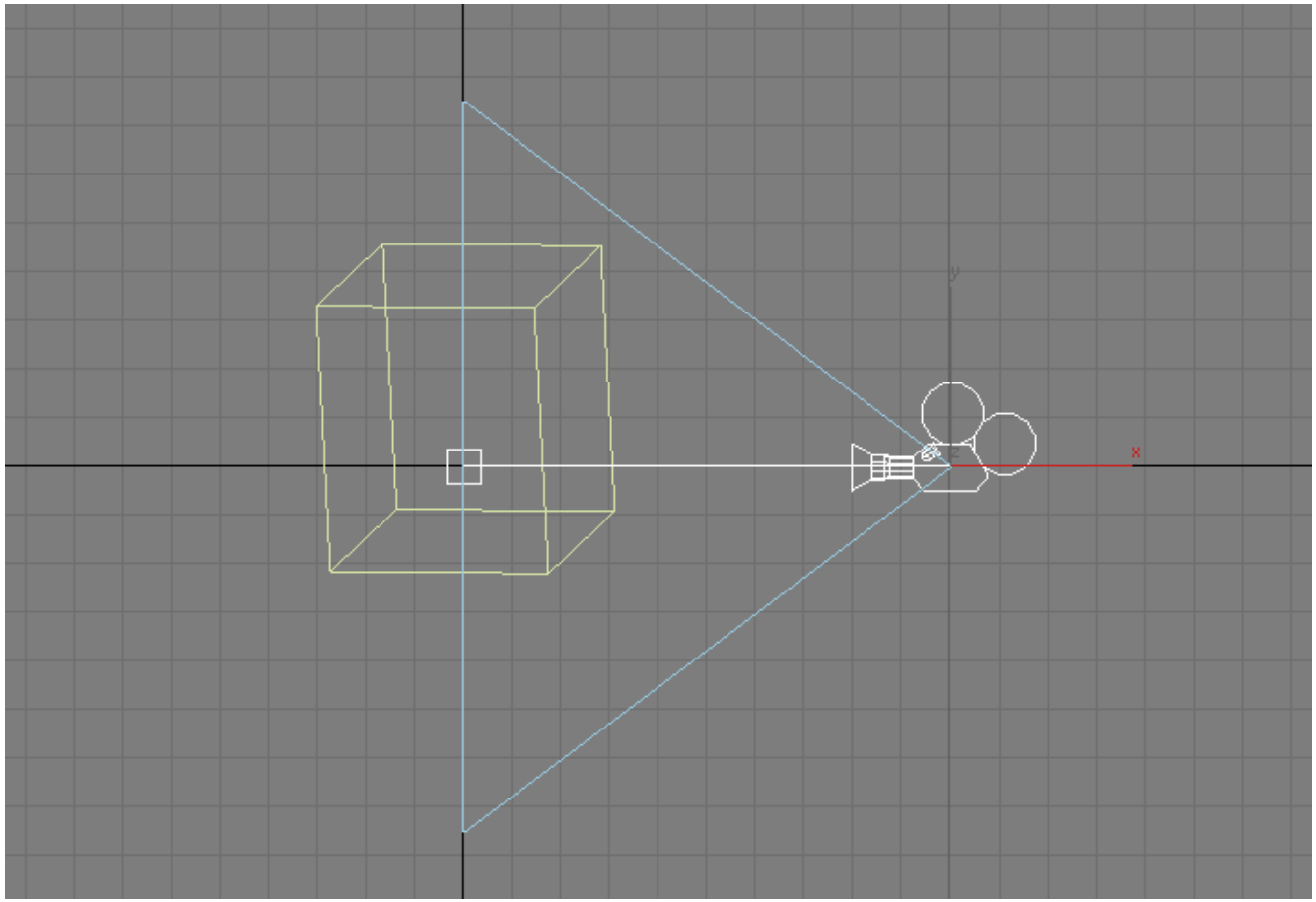
Определяем объект

- Вводим глобальную, “мировую” (world) систему координат (СК)
- Точка (vertex) – 3 координаты в МСК
  - struct Vertex { float x, y, z; }
- Грань (face) – 3 точки
  - struct Face { vertex[3] verts; }
- Это все, уже можно работать!
  - Только неэффективно (см. далее)

Определяем камеру

- Интуитивно – struct camera {
  vertex from;
  vertex to;
  float fov, aspect, roll; }
- Фактически – камера задает новую СК
  - X влево, Y вверх, Z из точки from в точку to
  - В коде работать удобнее с from/to/fov итп
  - Для отрисовки удобно задать матрицей

Простейшая камера



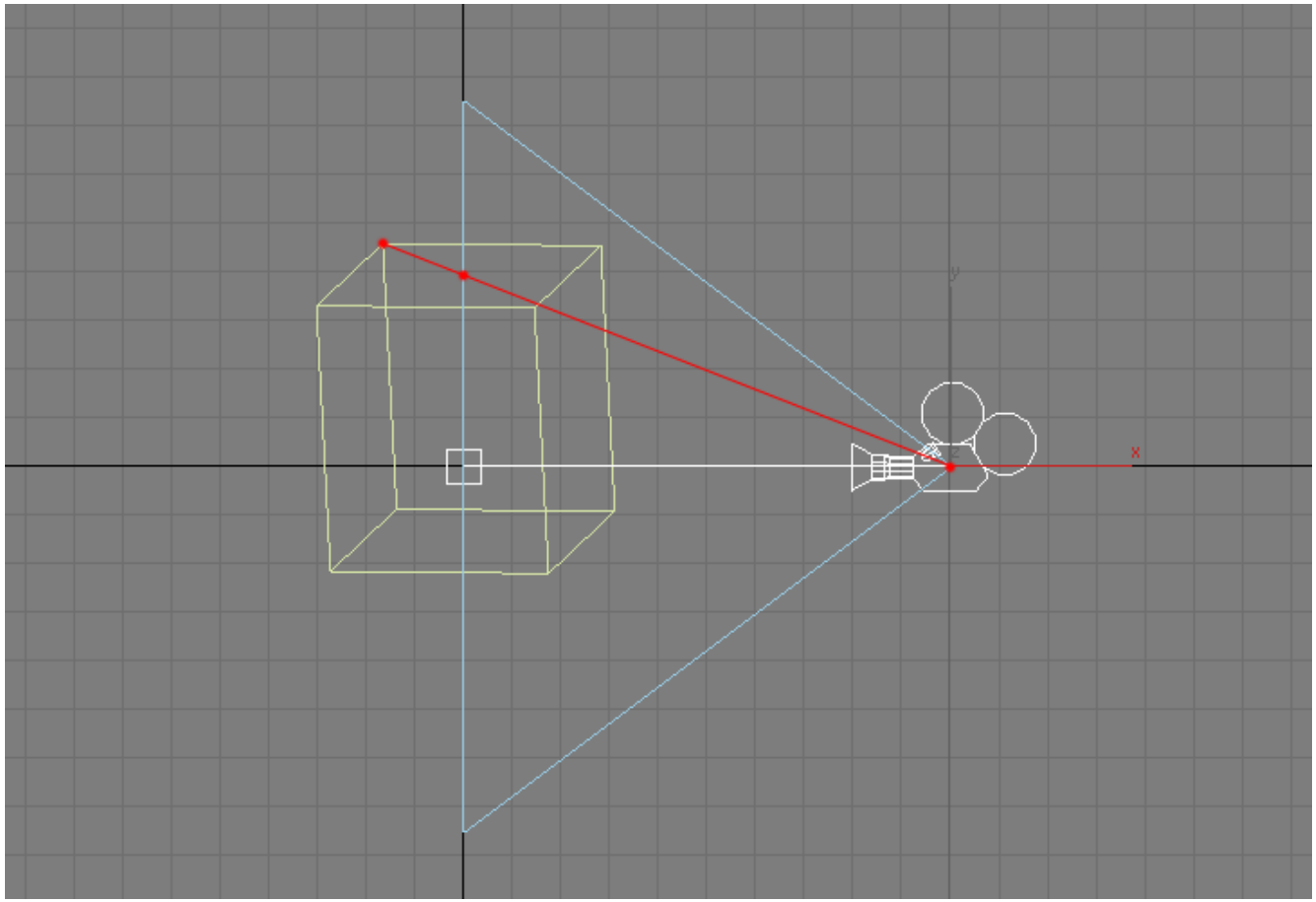
Application Developer Days

Простейшая камера

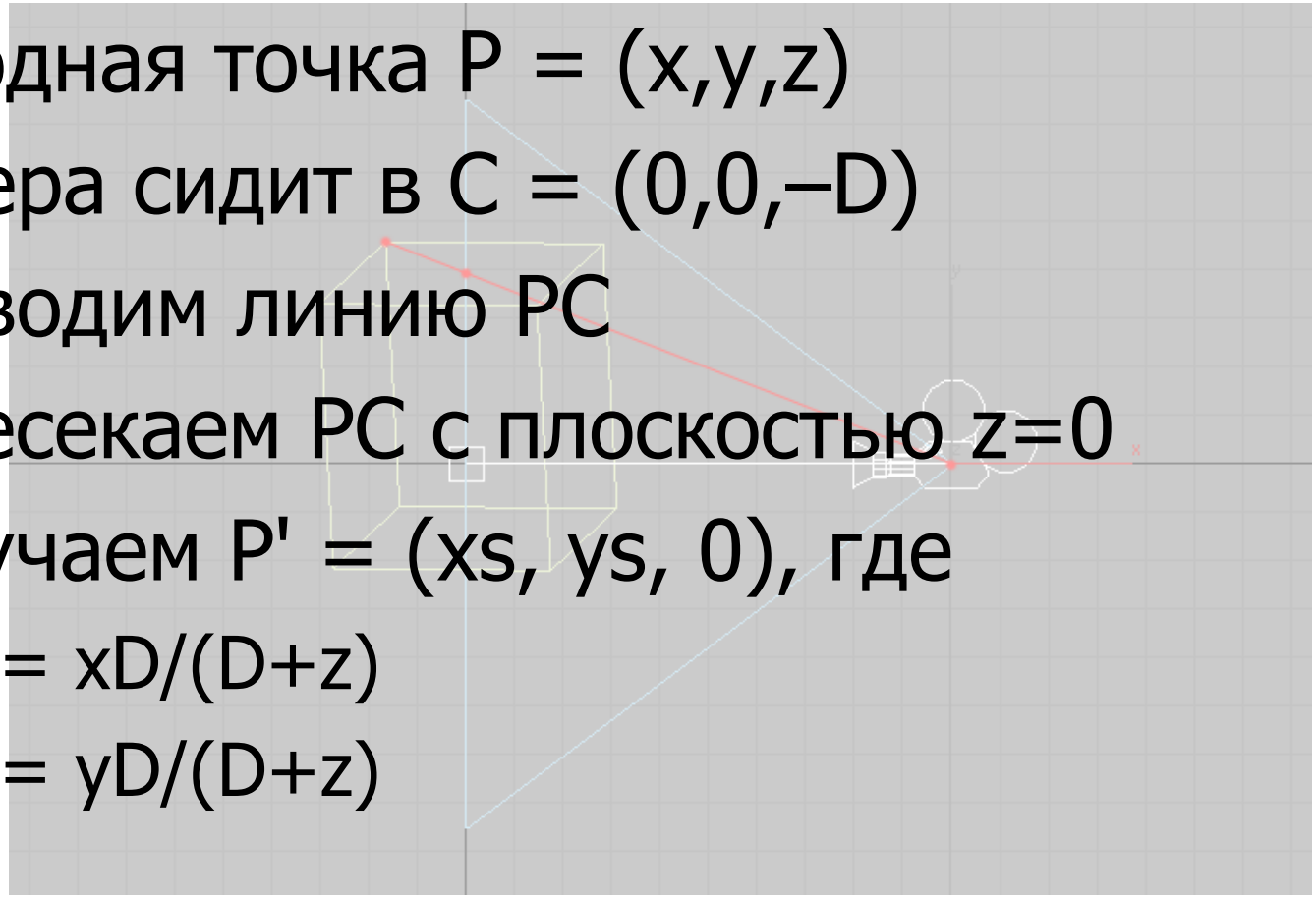
- Находится в (0, 0, –D)
- Смотрит в (0, 0, 0)
- FOV=90° → экран по X от –D до +D
- Aspect=1 → экран по Y от –D до +D
- Roll=0 → башка набок не повернута
- Перспективно проецируем каждую точку...

Application Developer Days

Проецируем...



Проецируем...



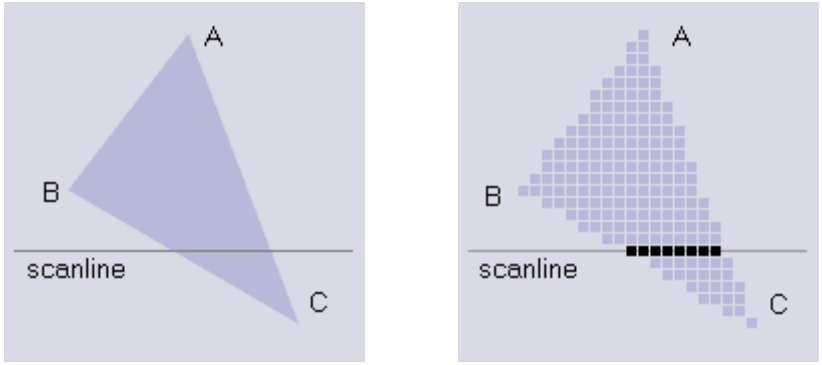
- Исходная точка P = (x,y,z)
- Камера сидит в C = (0,0,–D)
- Проводим линию PC
- Пересекаем PC с плоскостью z=0
- Получаем P' = (xs, ys, 0), где

  xs = xD/(D+z)

  ys = yD/(D+z)

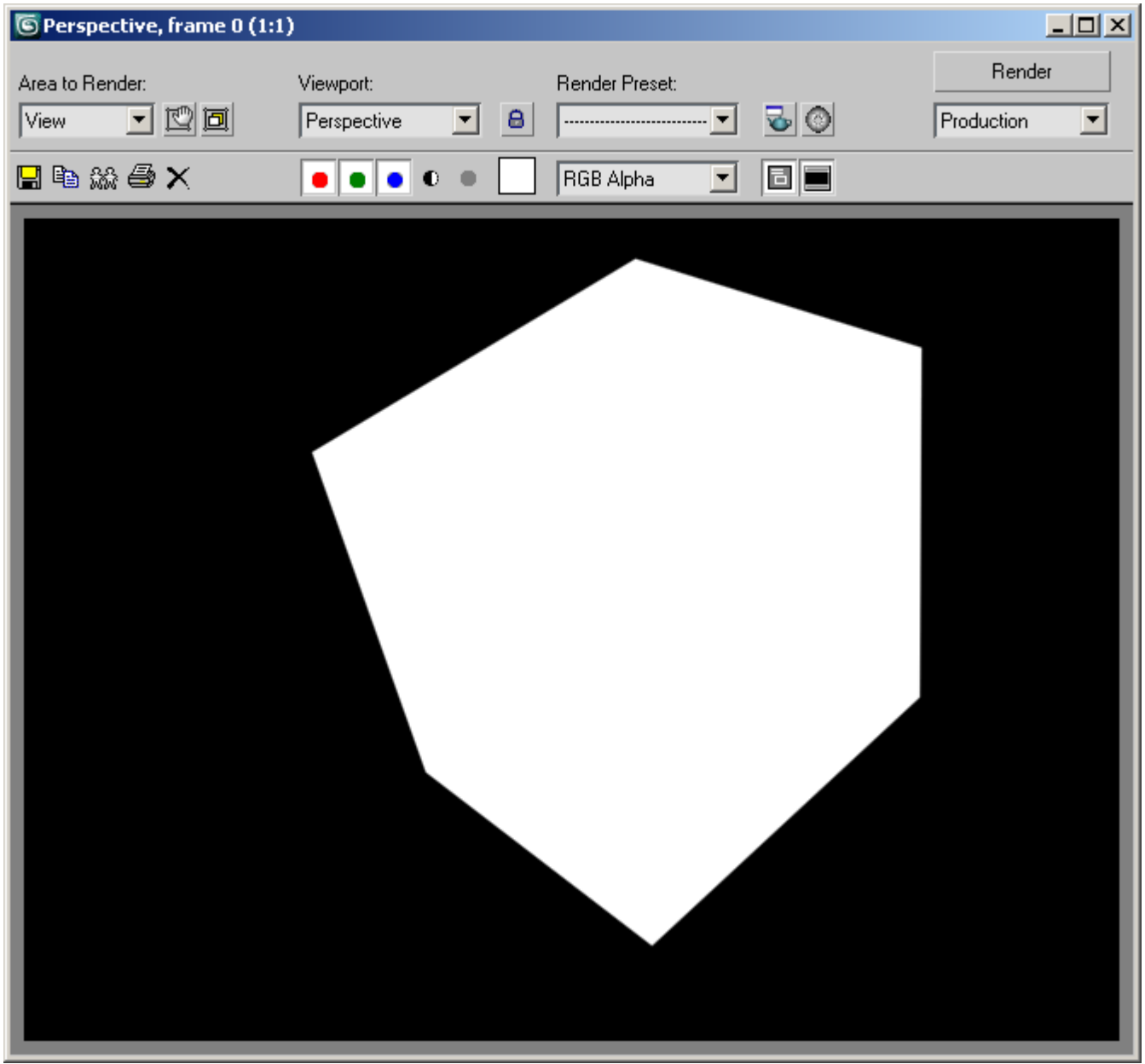
Рисуем...

- Проецируем 3 точки, рисуем треугольник на экране



- Повторяем 8 раз – победа, кубик взят!!!


Application Developer Days


Perspective, frame 0 (1:1)
Area to Render:
View
Viewport:
Perspective
Render Preset:
Render
Production
RGB Alpha

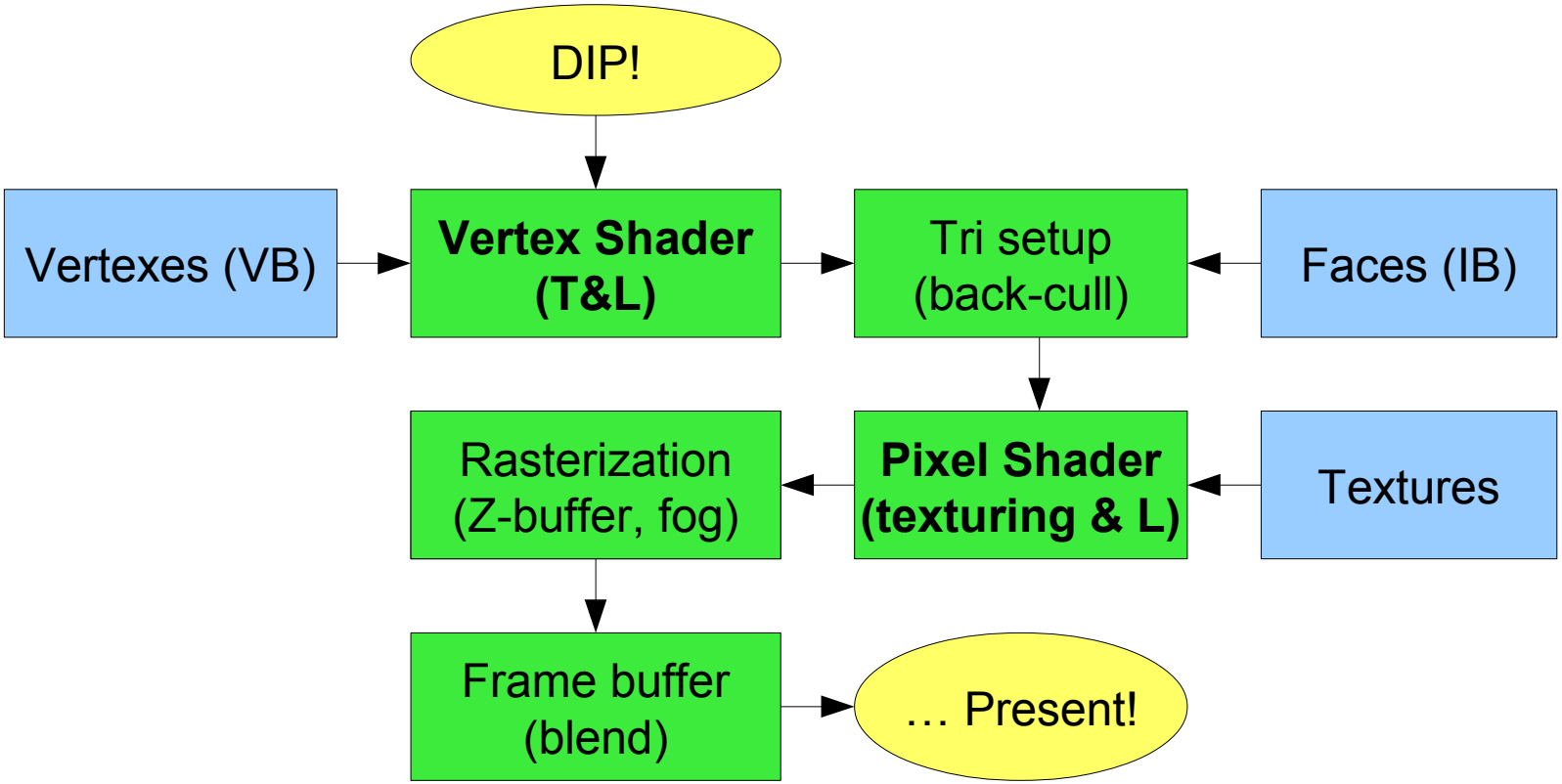
И, внезапно, куча новых проблем :)

- Нужно отсекать 2D tri об экран (viewport)
- Нужно отсекать 3D tri об znear
  - Иначе видим назад
- Нужно рисовать в “правильном” порядке
  - Которого... не существует
- Нужно исключить двойную работу (VB, IB)
- Нужно уметь двигать камеру
- Нужно что-то клевее тупой заливки белым!

Глупые проблемы решает железо

- Проецирование – делает GPU
- 2D отсечение – делает железо (GPU)
- 3D отсечение – делает GPU
- Порядок точек – делает GPU (z-buffer)
- Против 2х работы – GPU VB, IB
- Движение камеры – делает GPU
  - Как часть… проецирования

Конвейер GPU



Данные для GPU – в уставном виде

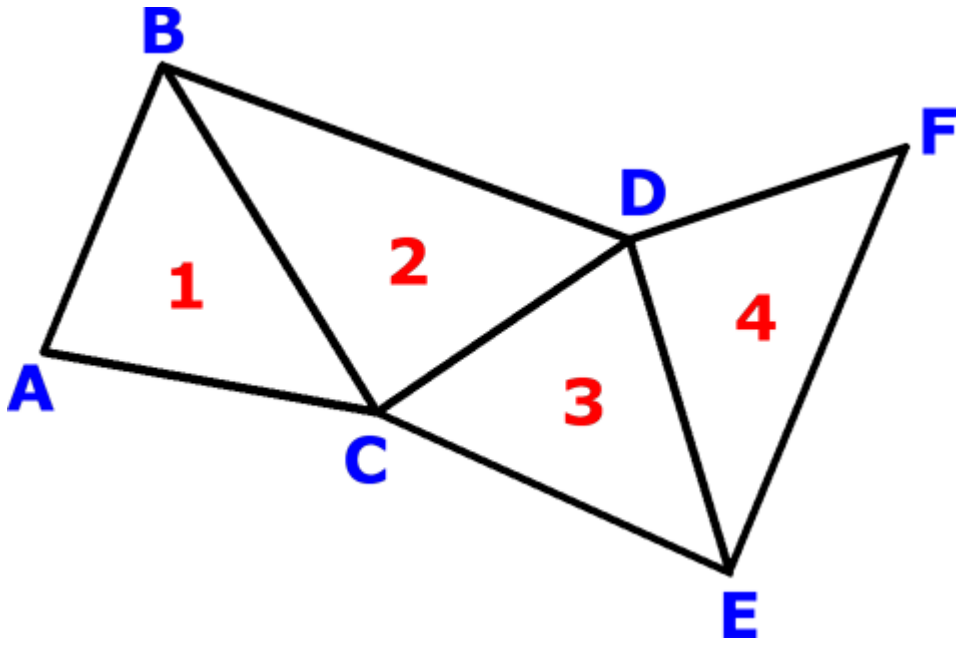
- 4 координаты вместо 3, добавляем w
- Ловко работаем с матрицами
  - Переносы “и так” задаются 3x3 матрицей
  - Повороты “и так” задаются 3x3 матрицей
  - Внезапно, теорема Шаля
  - 4x4 матрица дает еще и масштабировать
  - 4x4 матрица дает еще и проецировать
  - Итого 4x4 матрицы дают делать ваще все!!!

Данные для GPU – в уставном виде

- Вертекс это не только координаты
  - (несколько) пар UV для текстур (фактура)
  - Tangent space для карт нормалей (свет)
  - SH/BRDF/AO/... коэффициенты (свет)
  - Индексы и веса костей (анимация)
- Грани это индексы (!) вертексов
- Заливаются в идеале 1 раз (VB, IB)

Данные для GPU – в уставном виде

- Оптимизация граней, lists vs strips



Вы взрываете мне мозг!

- GPU принуждает и ограничивает
- Но, вообще говоря, к ПРАВИЛЬНОМУ
  - Матрицы, VB, IB, strips...
- Снимает ряд глупых головных болей
  - Больше не надо текстурировать ручками!!!
- Позволяет болеть качественно новыми
  - Свет, тени, анимация – и оптимизация!!!

Один (первый) кадр из жизни GPU

- Вливаем данные и код (*)
  - vertexes, faces, textures, V/P shaders
- Командуем ЗАЧИСТЬ (Clear)
- Тыщу раз подряд
  - Выбираем и биндим всякое
  - Командуем РИСУЙ (DrawIndexedPrimitive)
- Командуем ПОКЕЖЬ (Present)

  * – лучше только на “первом” кадре

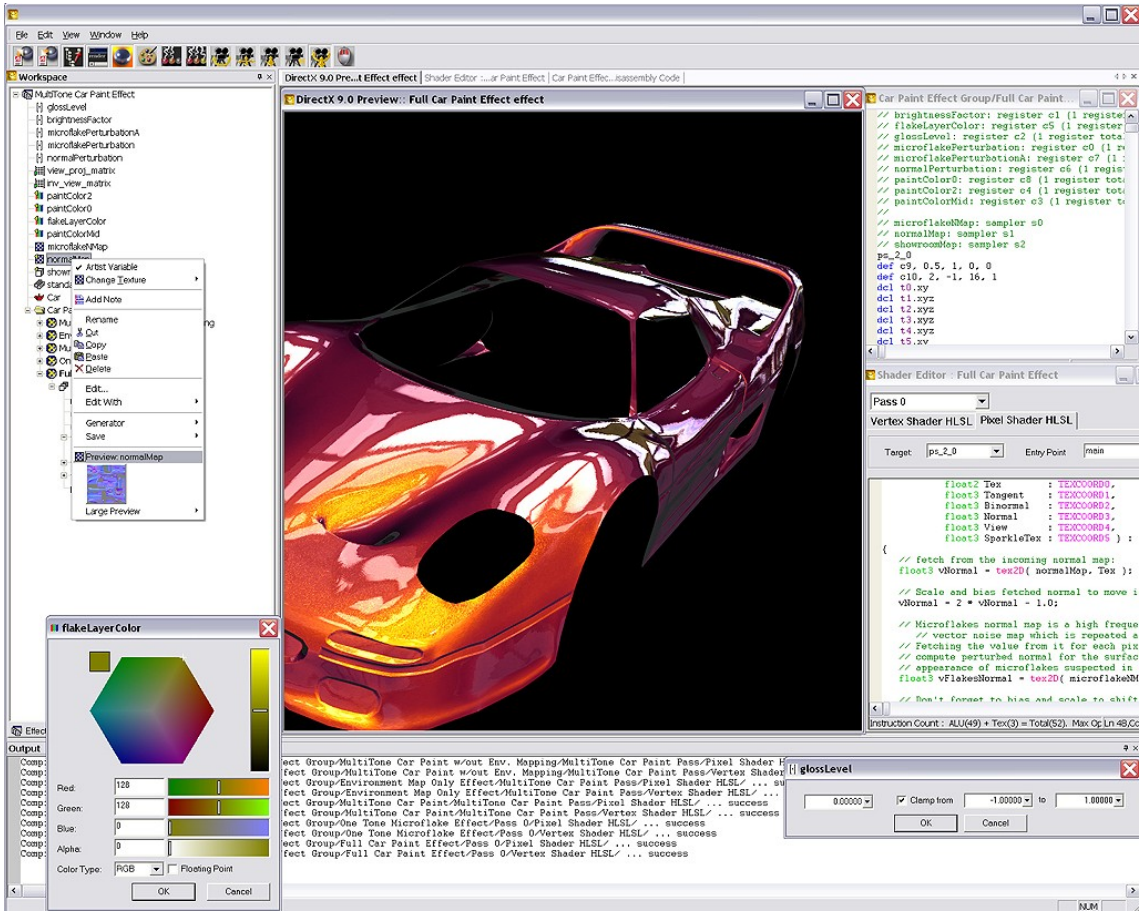
Как выглядит код?

```
// trivial HLSL vertex shader
VS_OUT sampleVS ( VS_IN In )
{
    VS_OUT Out;
    Out.Pos = mul ( In.Pos, matWorldViewProj );
    Out.BaseUV = In.BaseUV;
    return Out;
}

// trivial HLSL pixel shader
PS_OUT samplePS ( VS_OUT In )
{
    return tex2D ( smpBase, In.BaseUV.xy );
}
```

Application Developer Days

Куда его совать?!



What just happened?

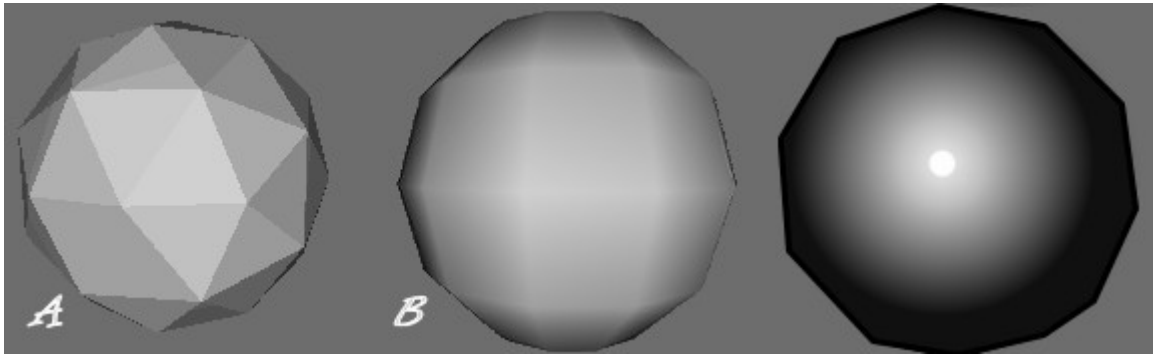
- Выборочный забег по вершкам?
- Ряд ключевых слов из математики и DX?

- **GPU программируется и нетяжело!!!**

- Пишем махонькие программки, “шейдеры”
- Они исполняются \any vertex, pixel соотв-но

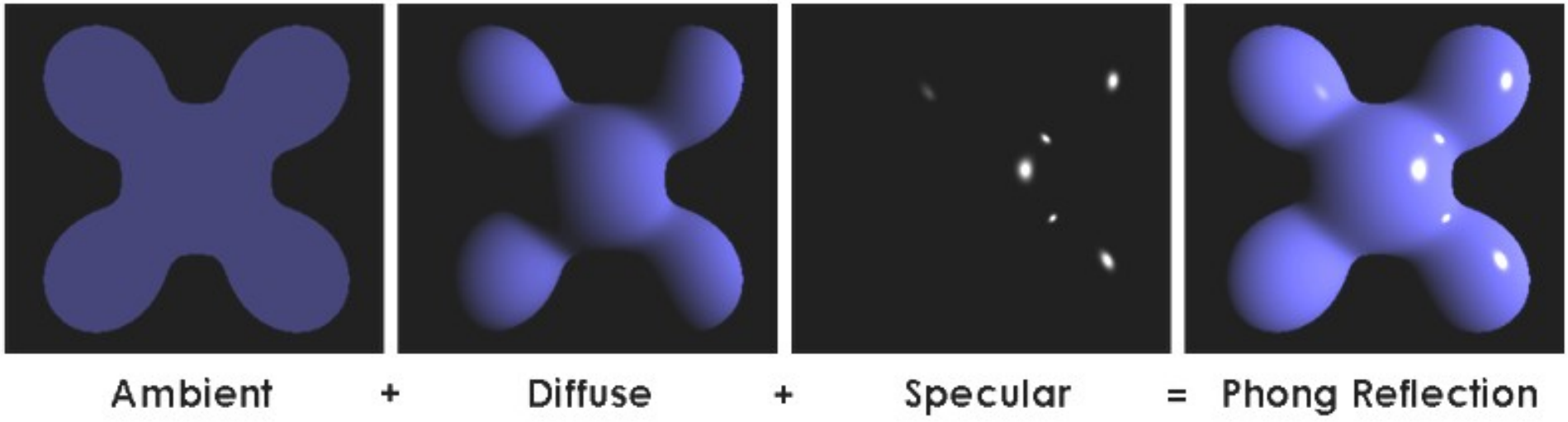
Да будет, пожалуй, свет

Классические модели освещения



- Flat = (FaceNormal dot L)
- Gouraud = Lerp(VertexNormal dot L)
- Phong = (N dot L) + (R dot V)^p,
  N = Lerp(VertexNormal)

Ambient/Diffuse/Specular...

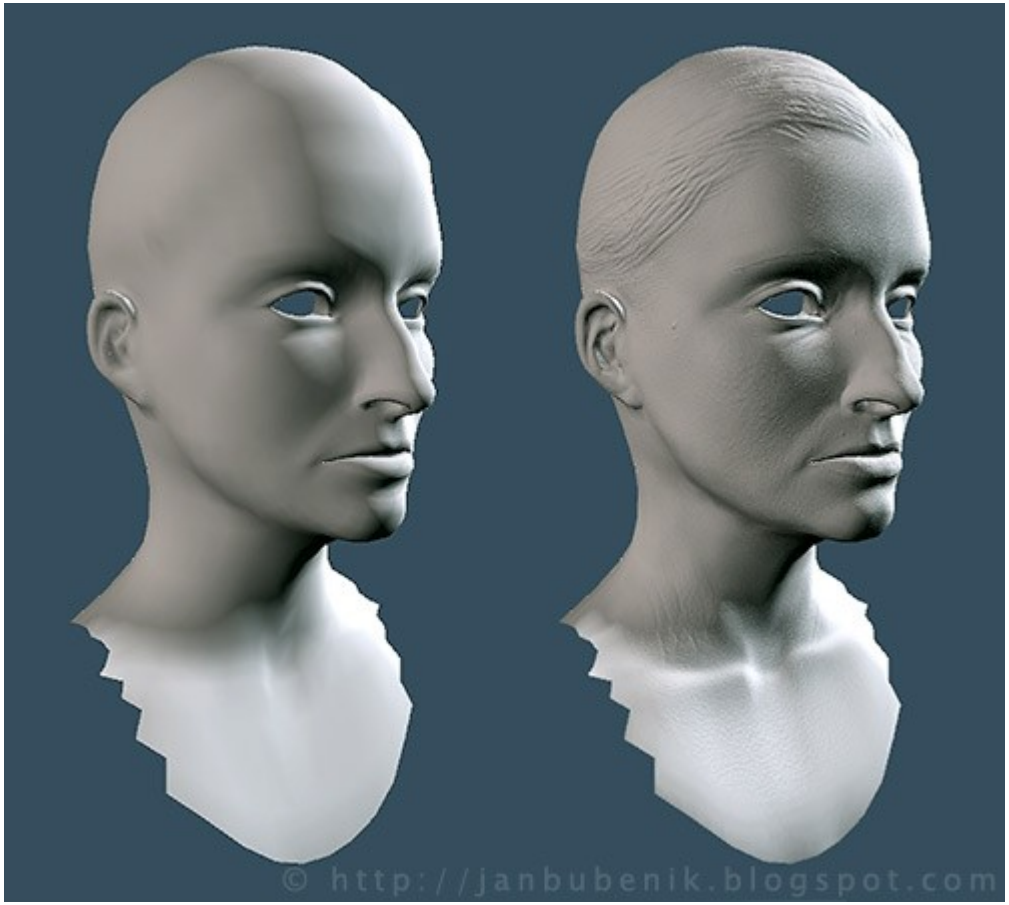


Diffuse

Diffuse + Bump

Diffuse + Bump + Specular

Нормаль это адово важно, например

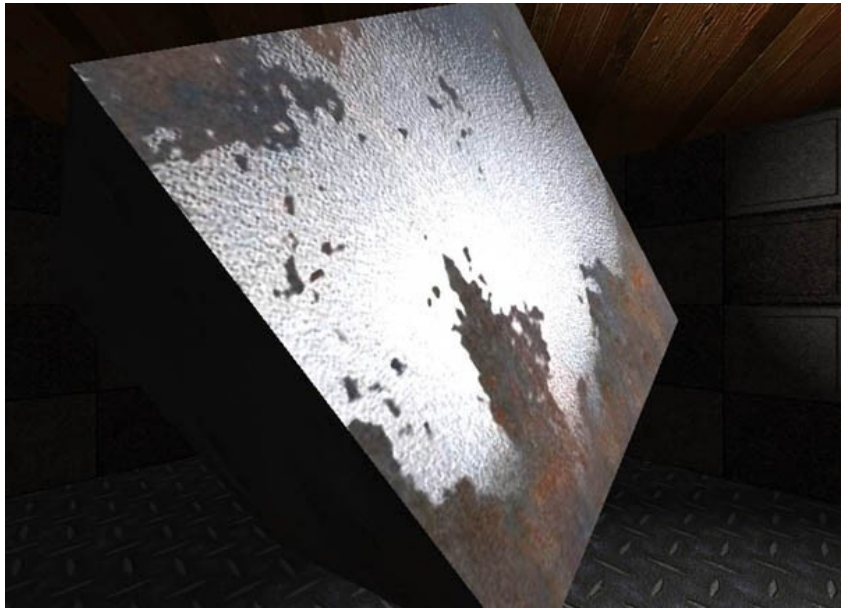


Мега-проблема

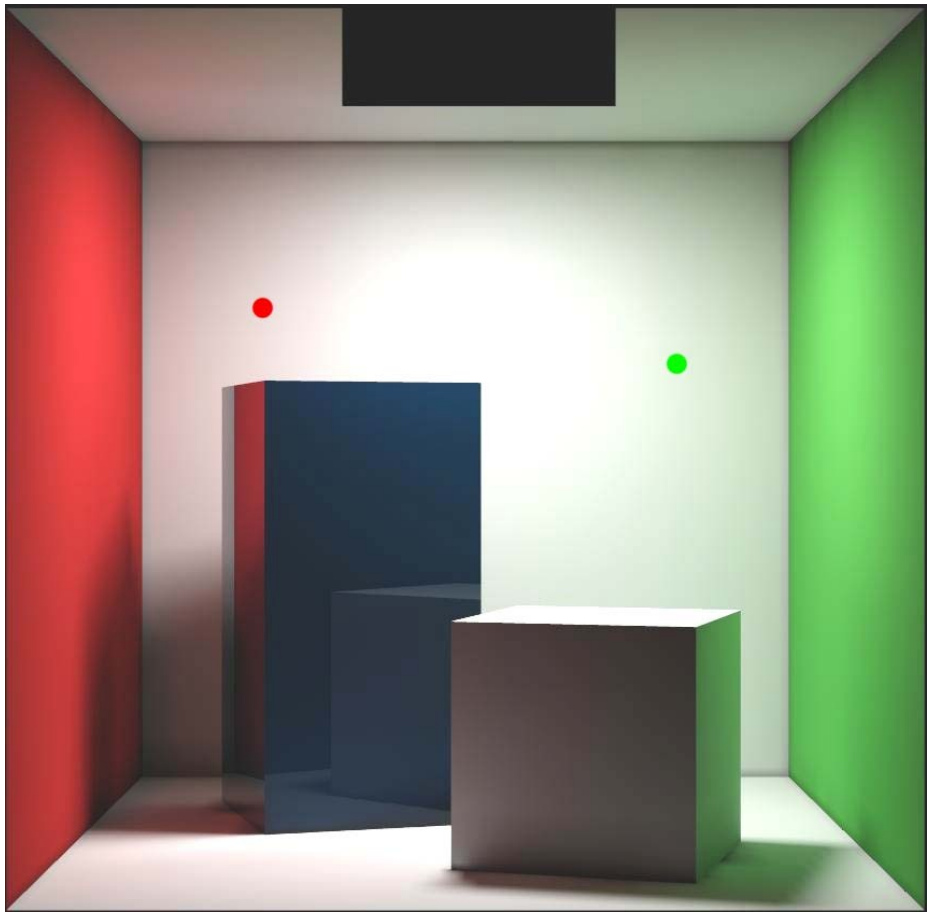
- Модель ambient/diffuse/specular простовата
- Свет СИЛЬНО сложнее
  - Неравномерность specularity, glossiness
  - **Непрямое освещение (indirect lighting)**
  - Анизотропия освещения (по самый BRDF)
  - SubSurface Scattering (кожа, итп)
  - Куча источников
  - Тени

Неравномерность spec/gloss

- Легко лечится текстурами
- Но, усложняет мега-уравнение
- s*(R,V)^p теперь берет s, p из текстур



Indirect lighting



Indirect lighting

- Без него плохо
  - Характерная картонная картинка
  - Унылые чорные тени
- Ambient плохое приближение
  - Унылые серые тени
- Считать его ТЯЖЕЛО
- Честный (!) расчет в RT – невозможен

Indirect lighting

- Для статики, lightmaps (еще одна текстура)

- Считаются заранее, оффлайн
- Только статика, не учитывает карты нормалей...

Lightmaps classic

Что же делать со светом...

- Ambient + Diffuse = смотрится плохо
- Lightmaps – идеально (!) но статично

- AO (Ambient occlusion), SSAO, PRT (Precomputed Radiance Tranfer) + SH (Spherical harmonics), Radiosity Bump, Diffuse cube lighting ... и еще куча техник

Что же делать со светом...

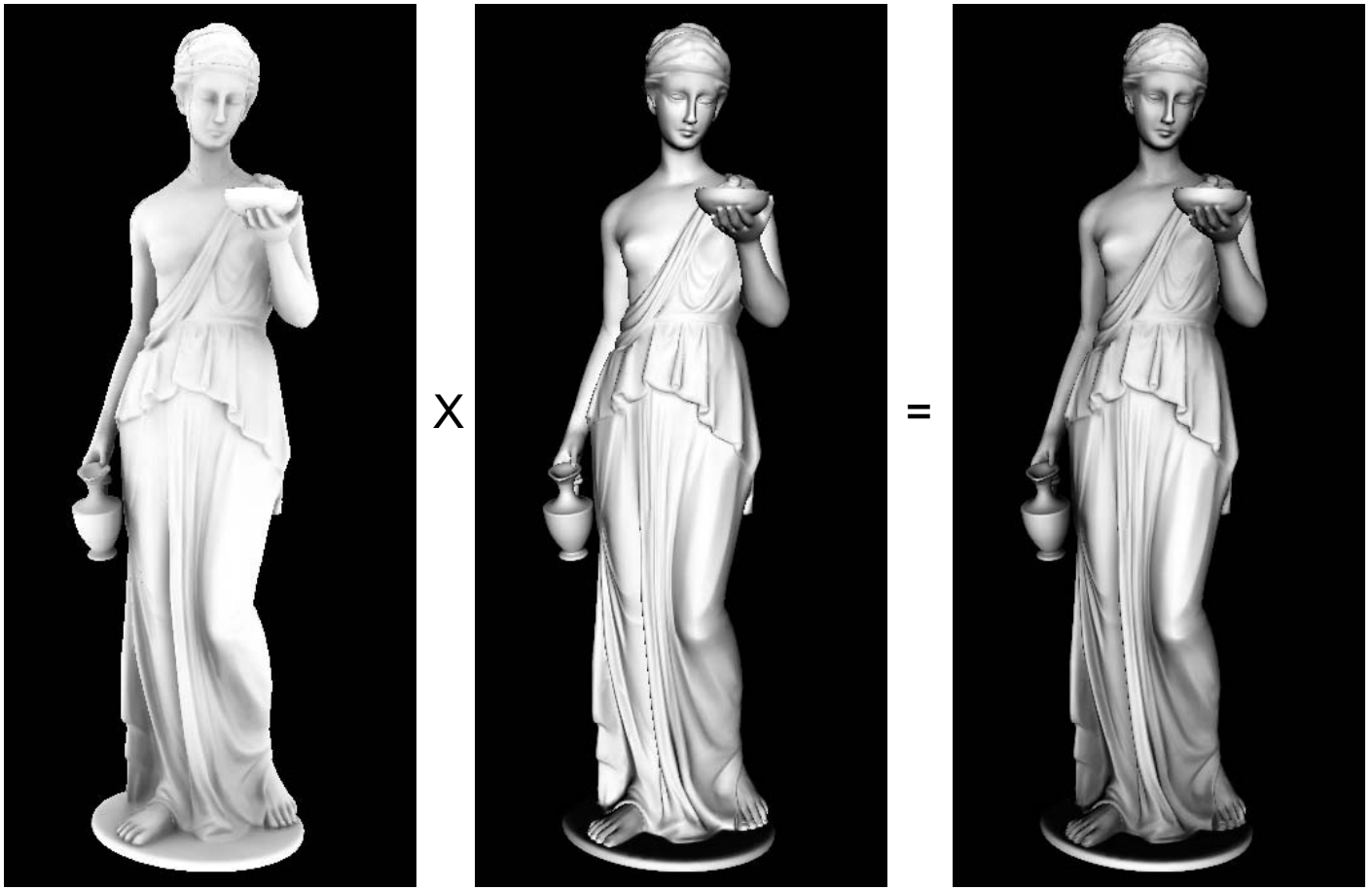
- Общий смысл всех техник освещения?

ДЕЛАТЬ КРАСИВО – И ПОХОЖЕ

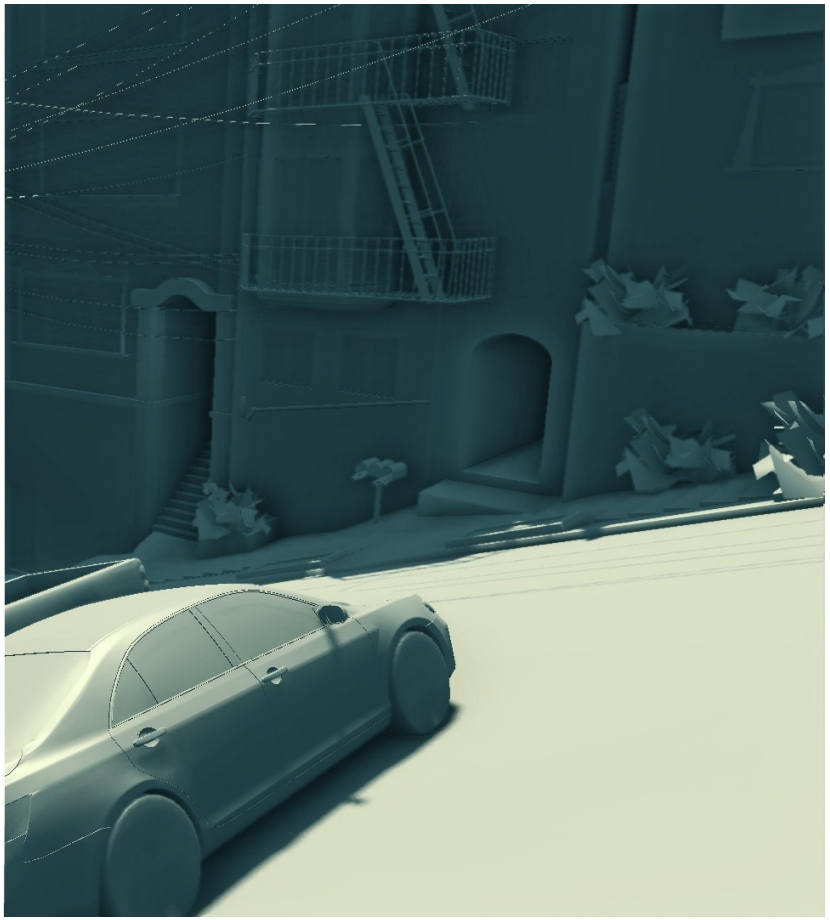
- Именно в таком порядке!
- Именно похоже, а не точно

AO

- “Сколько света долетает до этой точки?»

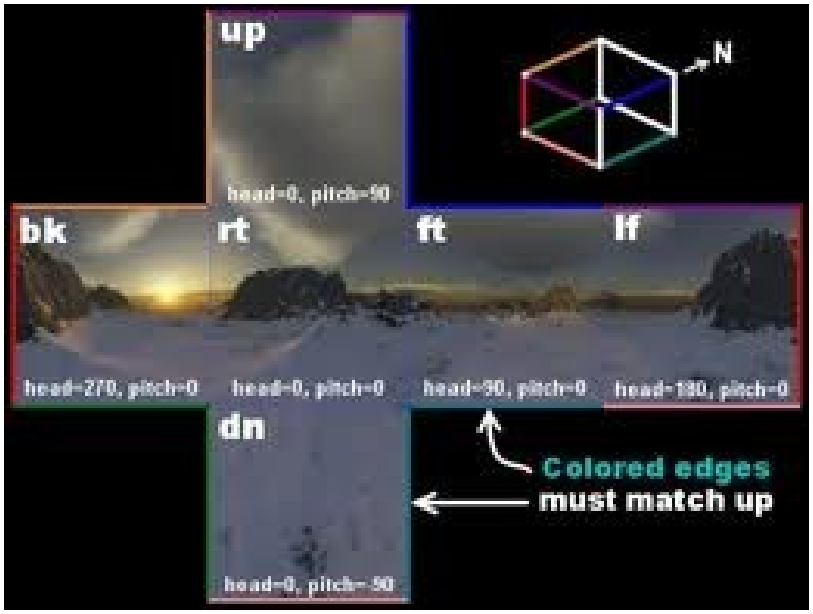


SSAO



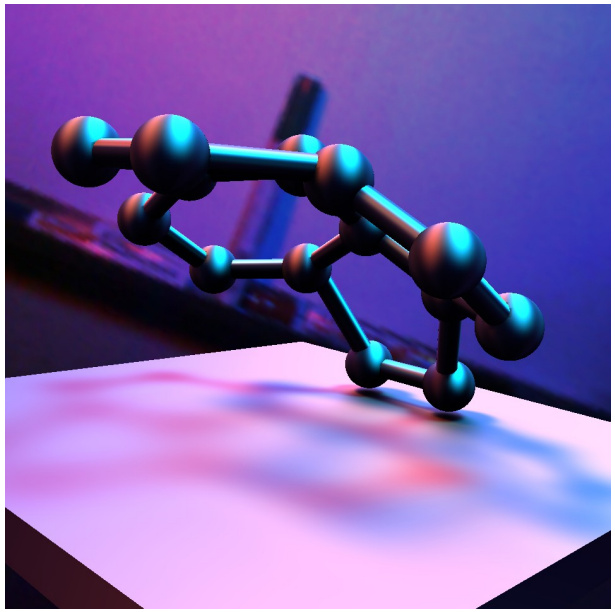
Diffuse cube

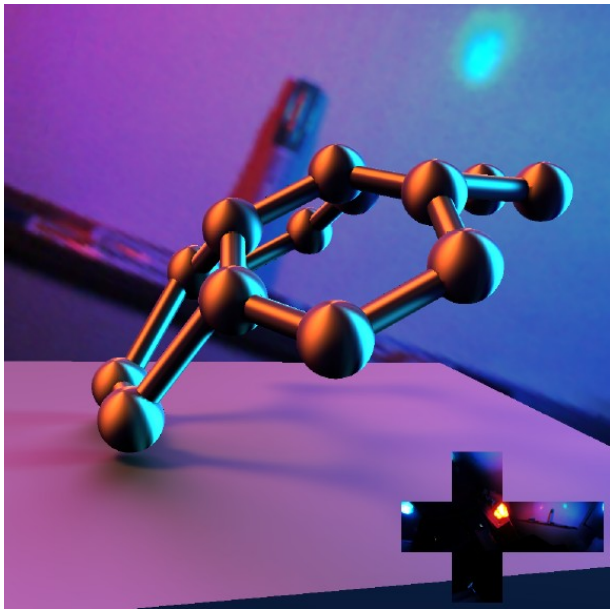
- Читаем Diffuse color из текстуры



“Объемные” источники – PRT, SH…

- Храним PRT (self-shadowing, inter-refl)
- Результат считаем из PRT+cube в динамике



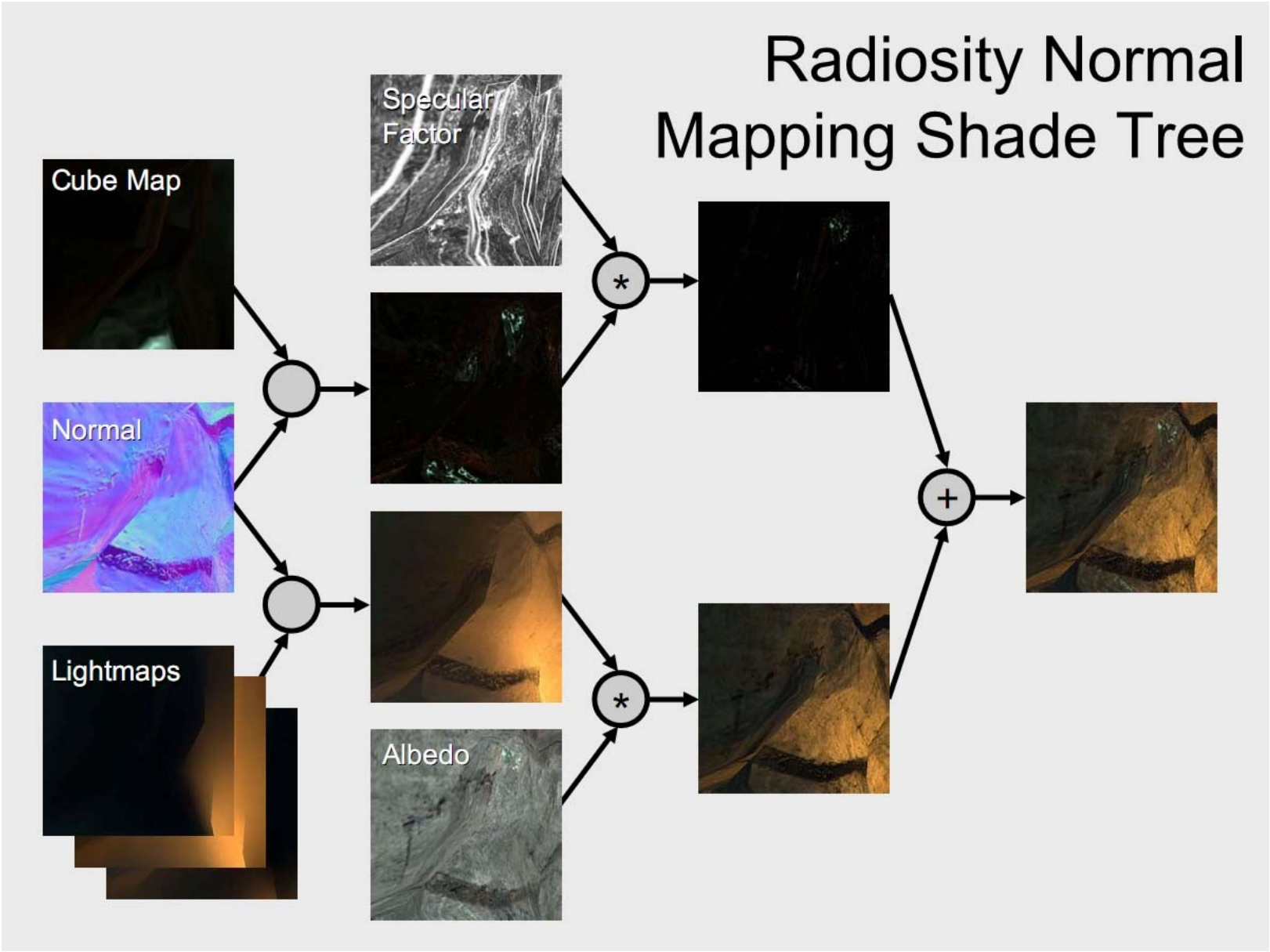


Radiosity normal mapping

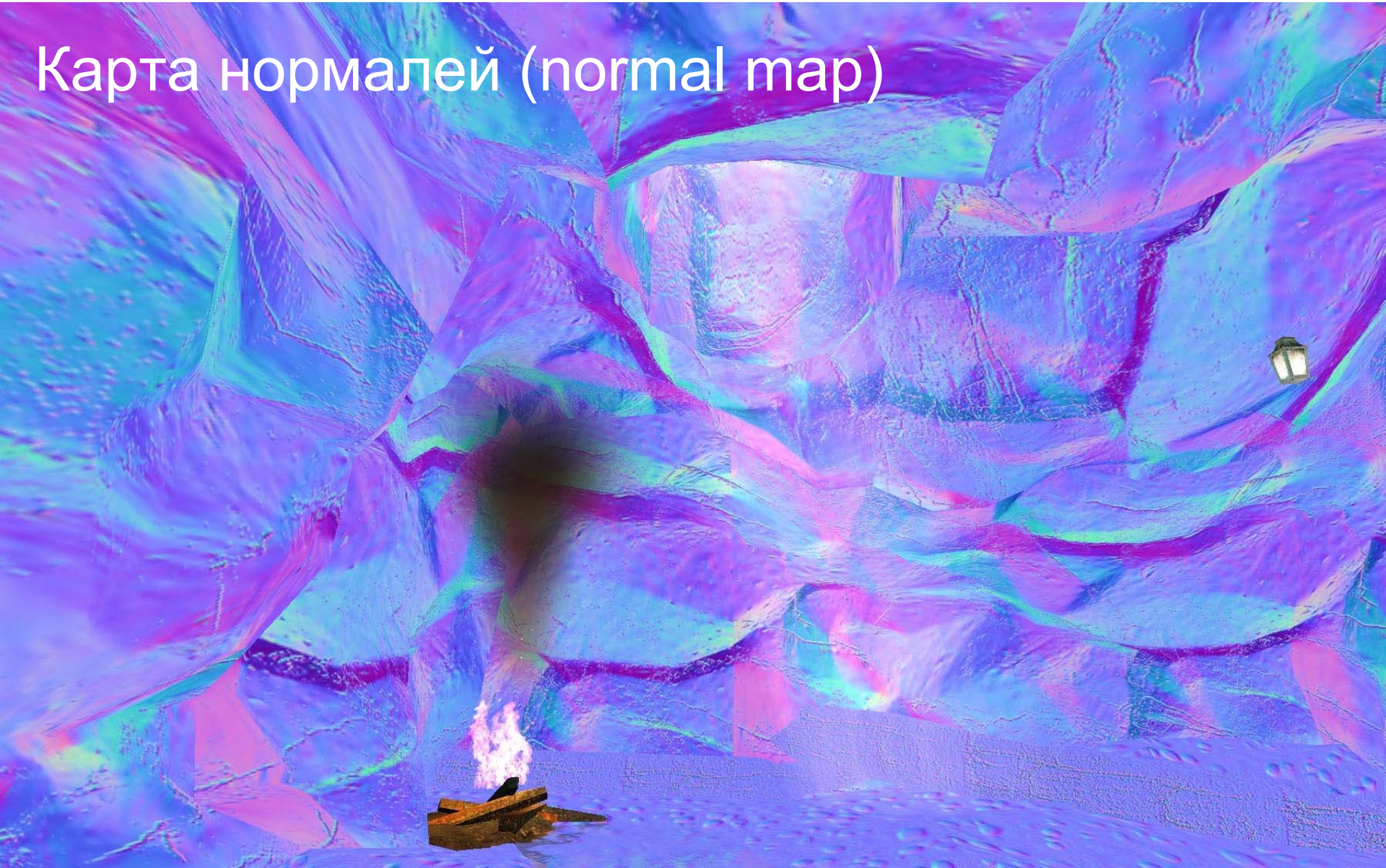
- И еще одна техника освещения
- А заодно, наконец-то боевой пример! (техника из Half-life 2, 2004)

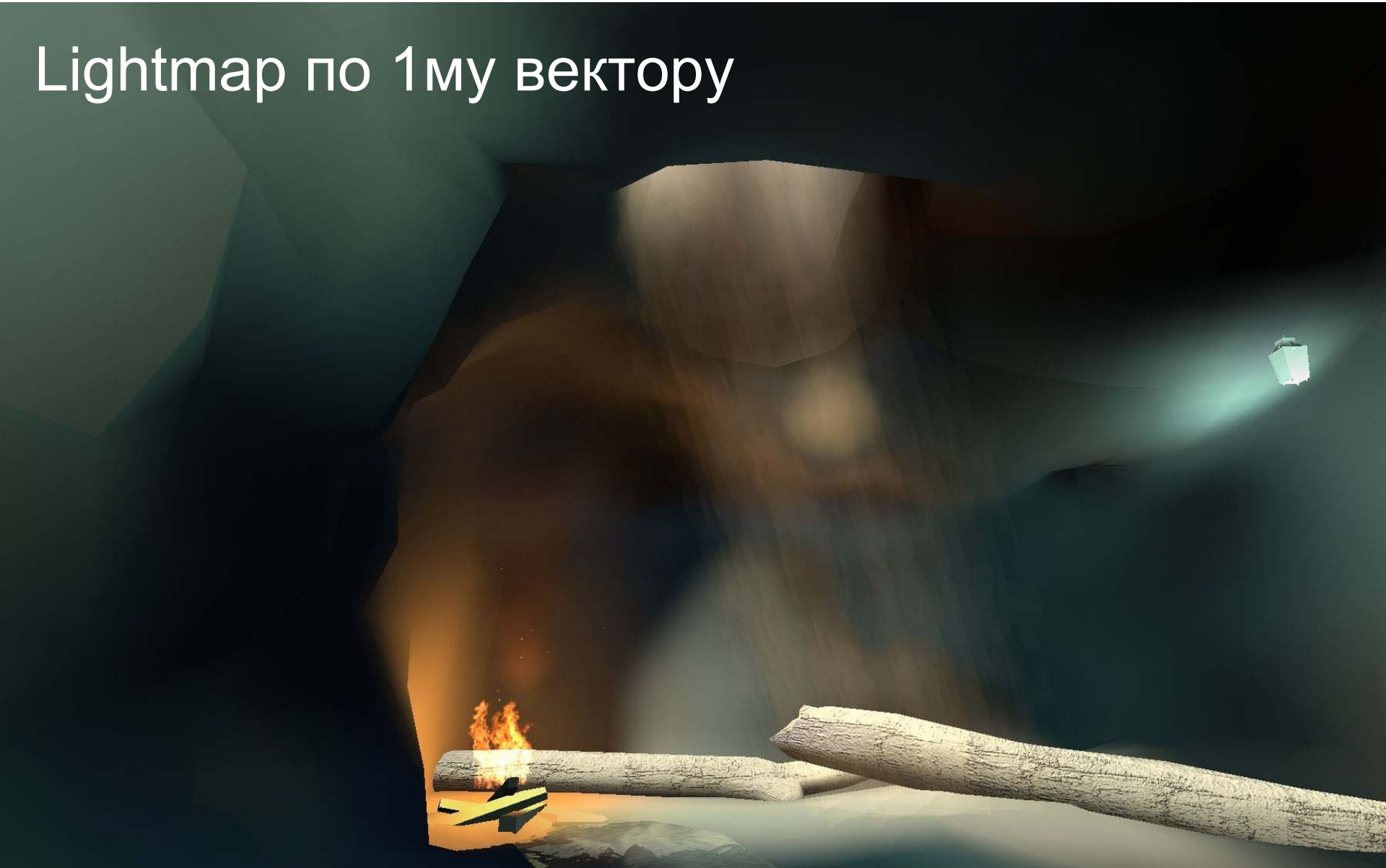
- Идея: храним не 1 лайтмапу, а 3
  - Свет, влетающий по разным направлениям
  - Чтобы уметь комбинировать с Normal mapping


Конечная цель


Radiosity Normal Mapping Shade Tree
Specular Factor
Cube Map
Normal
Lightmaps
Albedo
*
*
+

Обычный lightmap


Карта нормалей (normal map)


Lightmap по 1му вектору

Lightmap по 2му вектору

Lightmap по 3му вектору

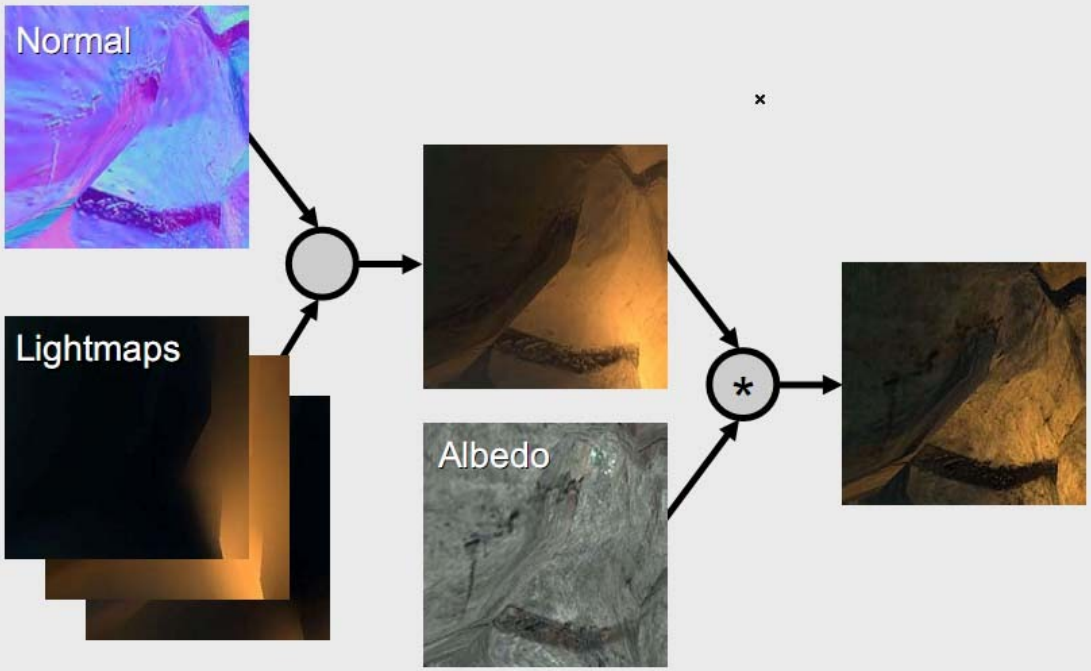
Nmap dot Lmap123 =
(Lmap1*Nmap.x + Lmap2*Nmap.y + ...)


Diffuse (редко albedo)

Diffuse * (Nmap dot Lmap123)

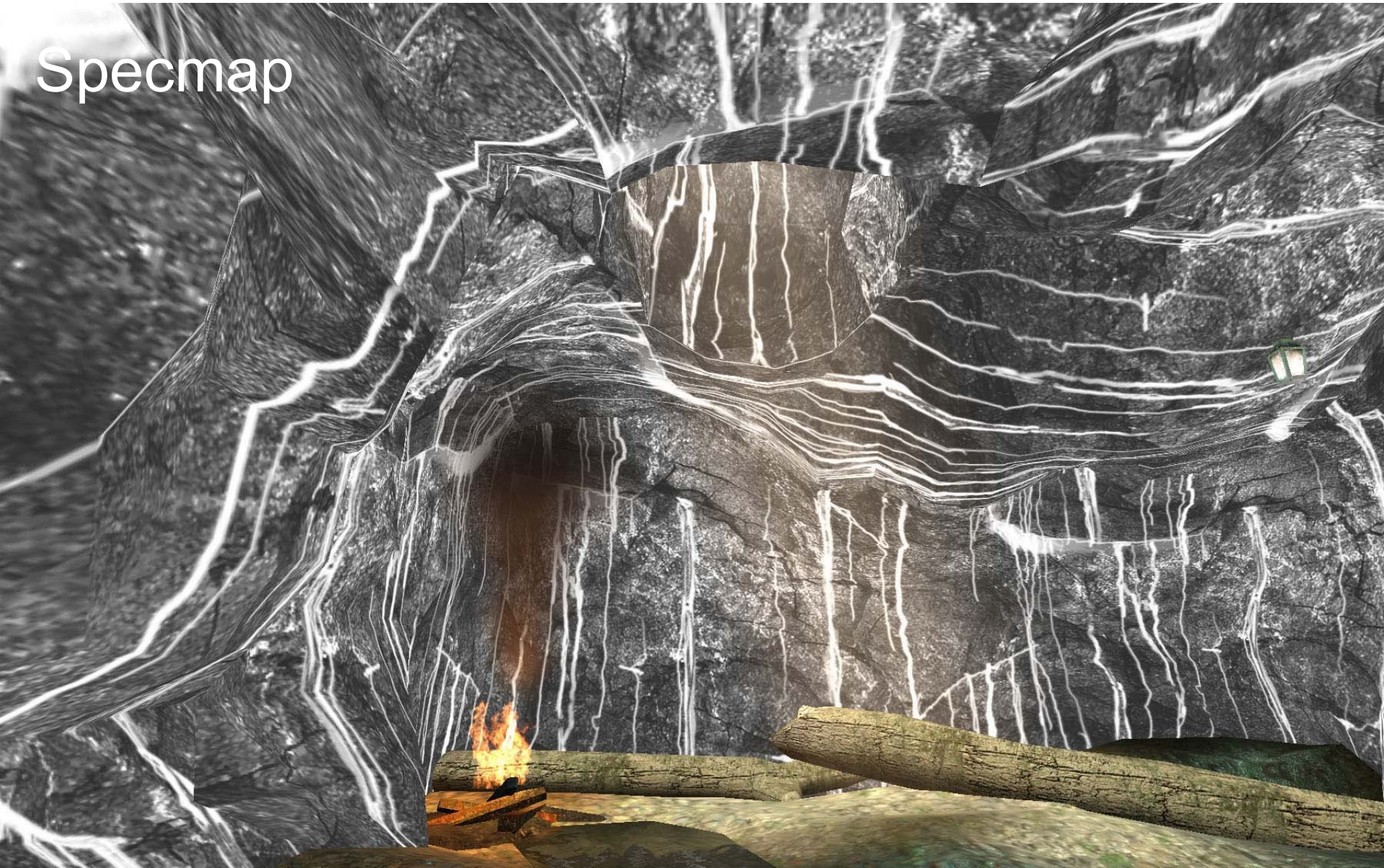
Diffuse * ПростоLightmap

Radiosity Normal Mapping Shade Tree



Cubemap (куча проб на уровне...)

tex(Cubemap, Nmap)


Specmap

Specmap * tex(Cubemap, Nmap)

FINAL (hl2)

Diffuse * Lightmap (quake3)

Боевые модели освещения

- Phong (1975) = Ka + Kd*DiffuseTex*(N,L) + Ks*SpecularTex*(R,V)^specPower
  - 1-4 текстуры: N, Diffuse, Specular, Gloss
- Q3 (1999) = DiffuseTexture * Lmap + Detail
- HL2 (2004) = DiffuseTexture * (Lmap1*N.x + Lmap2*N.y + Lmap3*N.z) + SpecularCubeTexture(N) * SpecularTexture

Свет это ВАЖНО

Свет это постоянные приближения

- Ряд ключевых выборов
  - 3 LM вместо 1 LM налицо
- Куча “мелких” улучшений
- Без каждого из которых – упс, уже не то и не так...

Железо как лимит

- Видеокарты толще – фокусов больше
- Q3, 1999 – 2-3 текстуры
- HL2, 2004 – 7 текстур
- ???, 2010 – страшно подумать!!!!

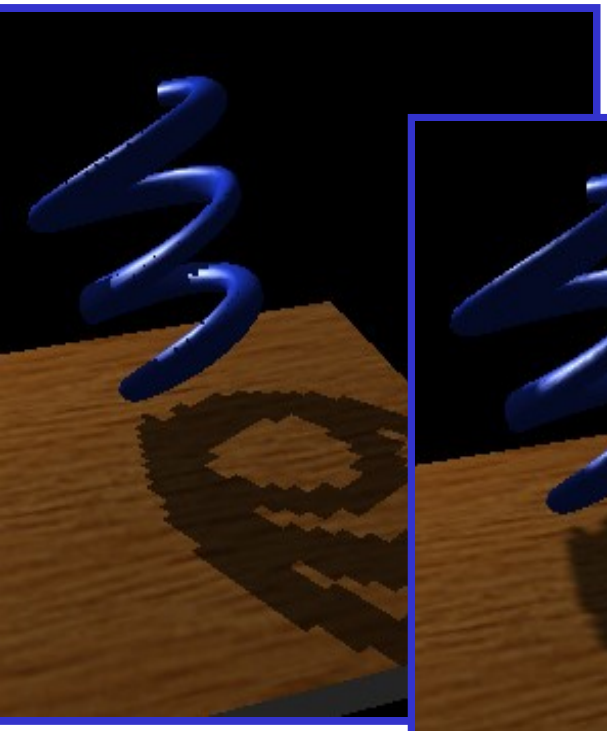
Что мы супер-кратко обсудили?

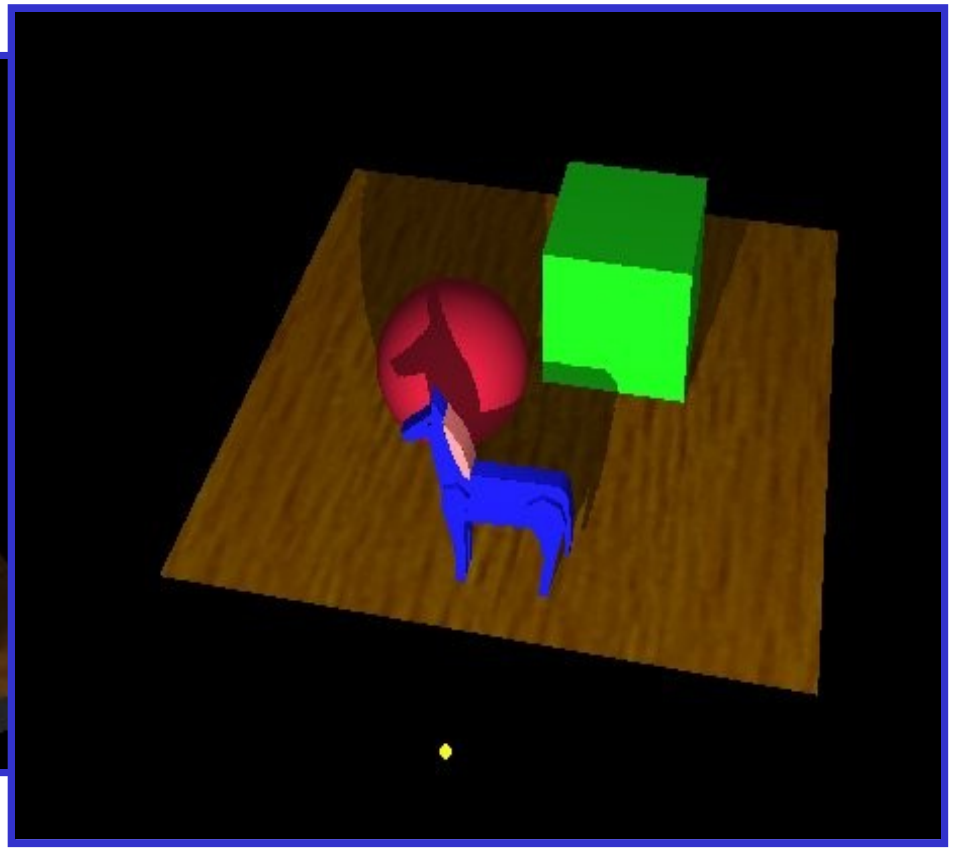
- Самые основы 3D (точки-камеры)
- Самые основы рендера
- Слегка посмотрели модели освещения
- Препарировали одну из моделей (хорошую!)

Что осталось?

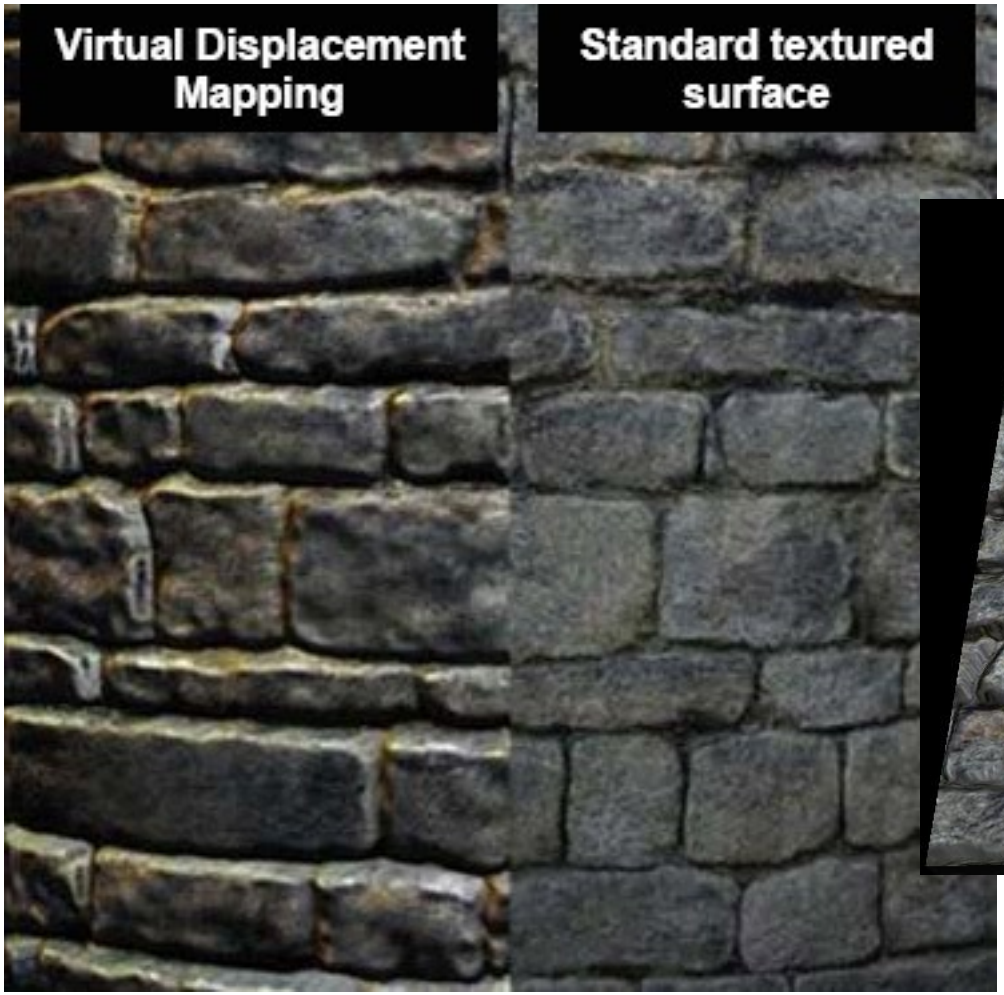
- Немного
- Shadow, parallax, displace mapping; skin, eye, hair, fur shading; reflection; refraction; transparency; motion blur; compression issues; particles; LODs; atmospheric effects; volumetric effects; post-processing; HDR...
- Vertex animation; skeleton animation, animation compression; inverse kinematics...

Shadow mapping





Parallax mapping



Parallax mapping

Texture Mapped

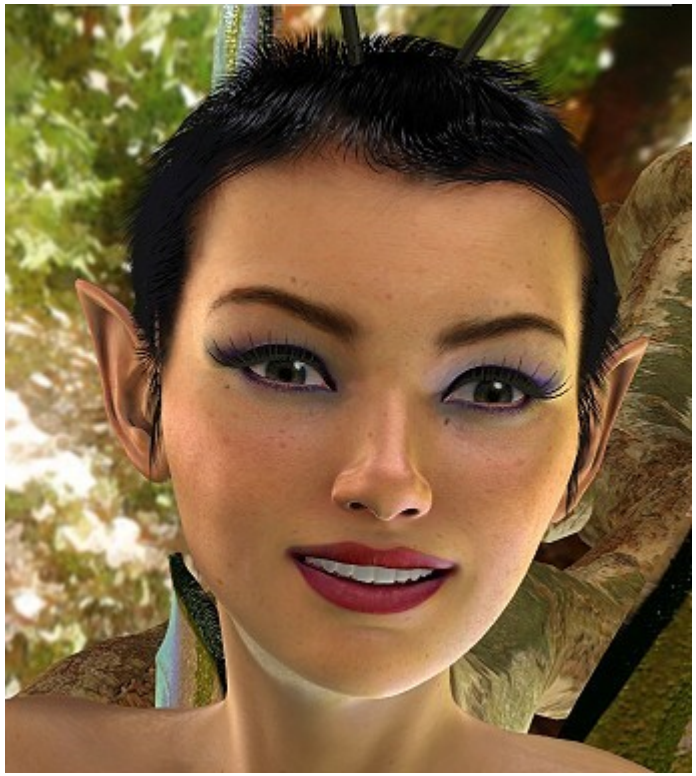
Normal Mapped

Parallax mapping

Parallax Mapped

Steep Parallax Mapped

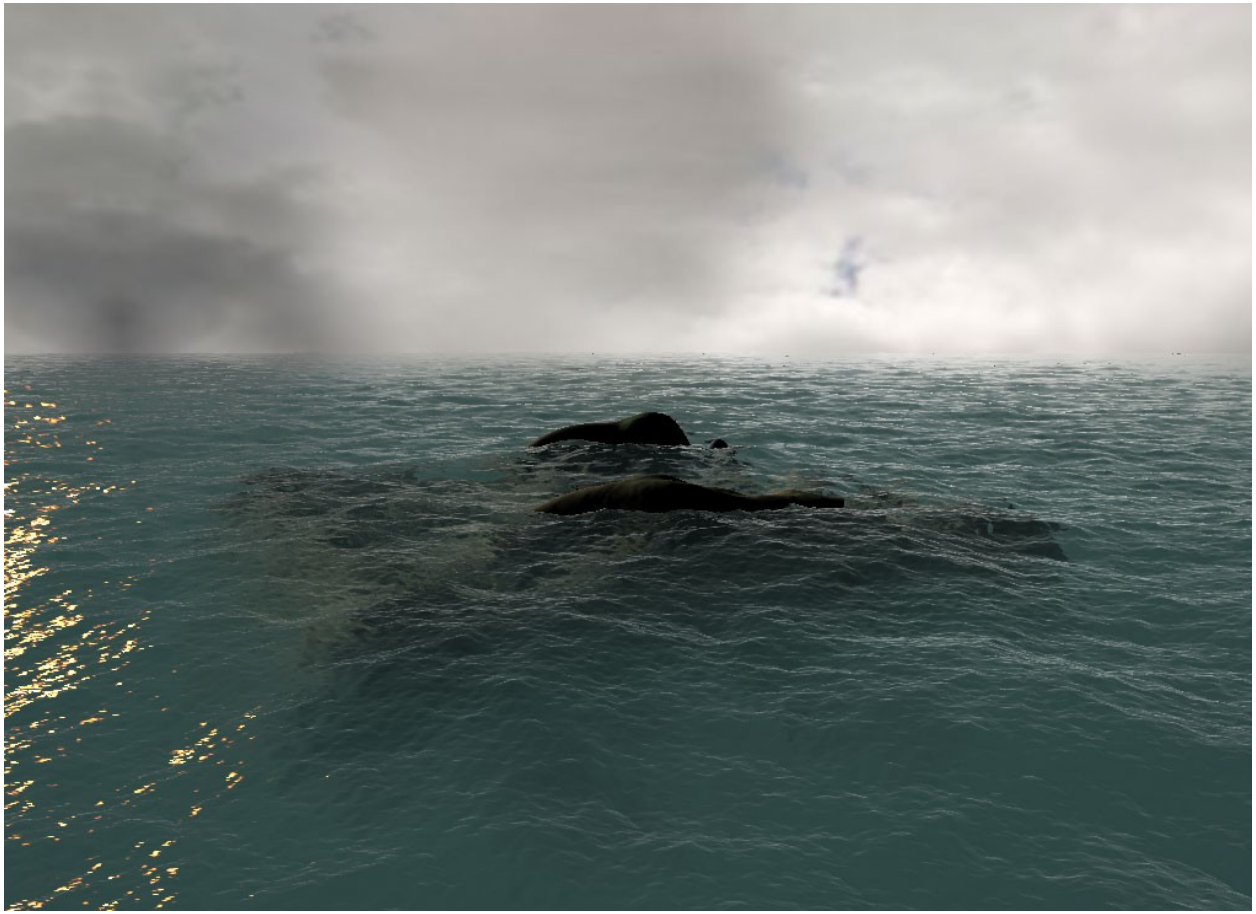
Skin/eye/hair shading



Fur shading

Water

Atmospheric effects



Анимация...

- Дзен-слайд, исполняется жестикуляцией.

Application Developer Days

Итого

- Графика – это очень просто!
  - Кубик нарисовать легко!
- Графика – это сложно
  - МНОГО всяческих техник
  - Важно выбрать правильные
  - Важно суметь сделать
- Главное – не увлекаться физикой.

Application Developer Days

ВСЁ

(а завтра будет скучное занудство без клевых картинок)

3 измерения на 3 пальцах // Андрей Аксенов // ADD 2010


Application Developer Days