Современные графические технологии

или

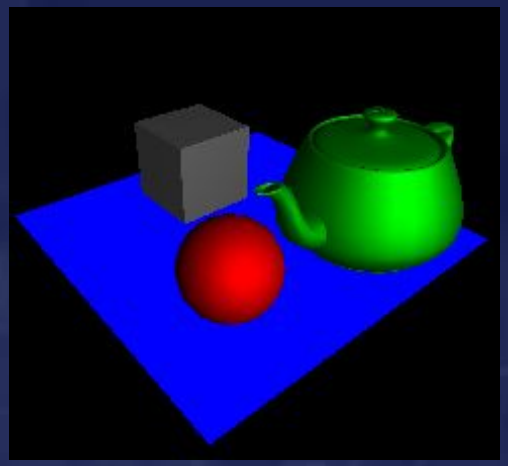
OpenGL и графические процессоры

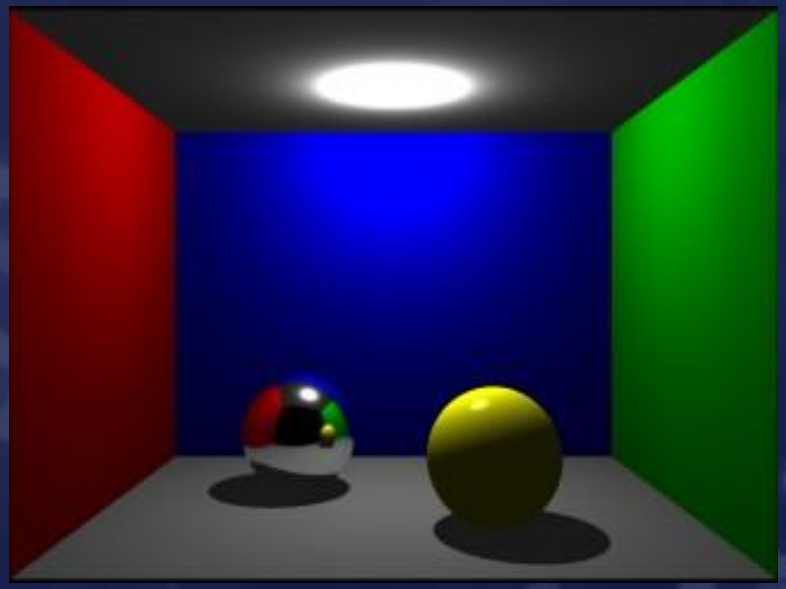
2010

Методы создания изображений

Точность и реалистичность:

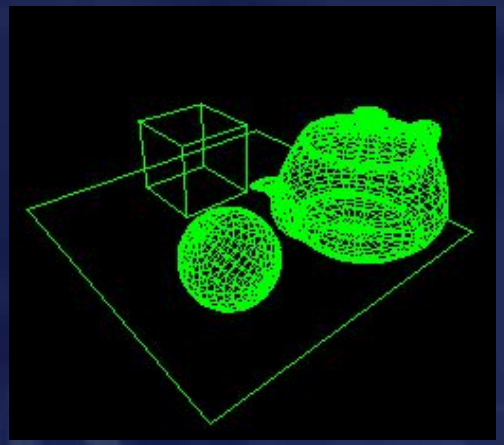
- Трассировка лучей
- Излучательность
- Фотонные карты





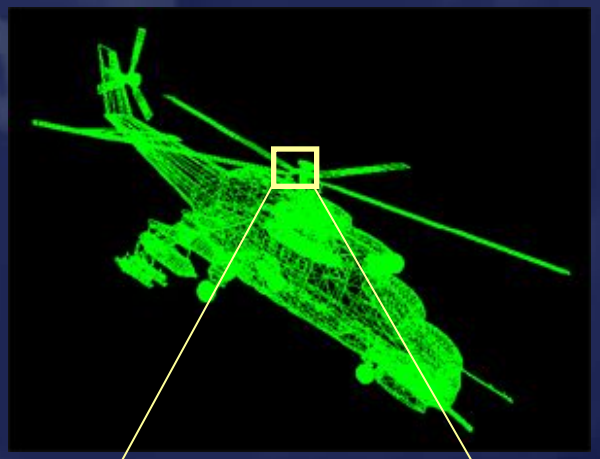
Скорость:

- Полигональная графика



Полигональное представление объектов

Объект задан набором вершин, которые объединены в плоские грани, чаще всего – треугольные.



Для каждой вершины заданы:

- Координаты вершины
- Нормаль
- Координаты текстуры
- И много чего еще ...


Menu
Animation
Emotions
Rendering
About
Help
Quit
NVIDIA


COLORS


Menu
Animation
Emotions
Rendering
About
Help
Quit
15
GPU:576MHz MEM:900MHz GPUt:83C
OpenGL SMU:194Mb
NVIDIA

GPU vs. CPU

GF100

RV870

- > 3 млрд. транзисторов
- Тактовая частота 700Mhz
- 1.5GB GDDR5 памяти

- ??? транзисторов
- Тактовая частота 825Mhz
- 1-2GB 1300 MHz памяти

- Тактовая частота 3.3Ghz
- 1.17 млрд. транзисторов (six core)

Core i7-980X

GPU vs. CPU (4 года назад)

NV30

- 120 млн. транзисторов
- Тактовая частота 500Mhz
- 128MB 500MHz памяти

R300

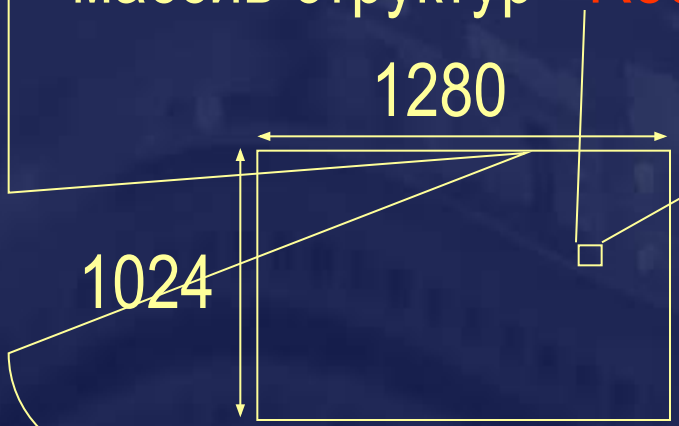
- 107 млн. транзисторов
- Тактовая частота 325Mhz
- 128MB 310MHz памяти

- Тактовая частота 1.6Ghz – 3.06Ghz
- 42 млн. Транзисторов (core)

Архитектура GF100

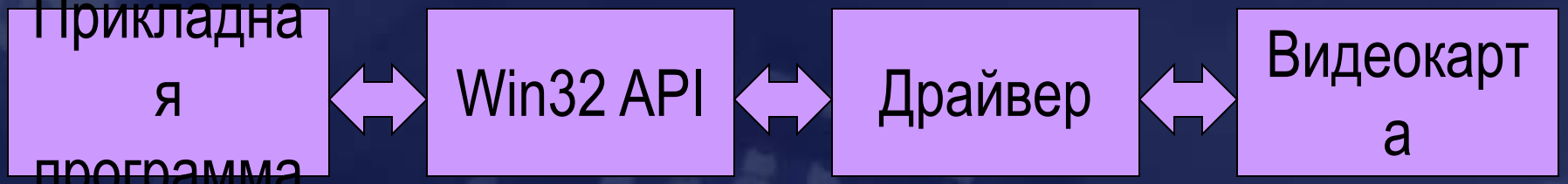
Буфер кадра

Буфер кадра – прямоугольный массив структур <Red,Green,Blue>

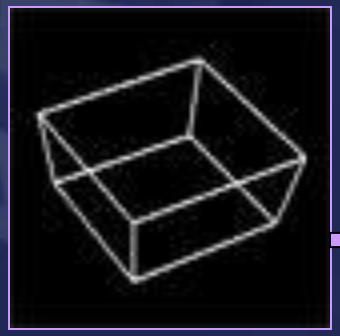


2D-ускорители

- Копирование и перемещение прямоугольных блоков
- Масштабирование прямоугольных блоков
- Отрисовка курсора мыши
- Отрисовка прямых линий и других примитивов



Графический конвейер



$V_i=\{P,n,\ldots\}$

$F_j=\{V_1,V_2,V_3\}$

T&L:
Преобразование
и освещение

$V_i'=\{P',RGBA,\ldots\}$

$F_j'=\{V_1,V_2,V_3\}$

Rasterization:
Разбиение
примитивов
на пиксели

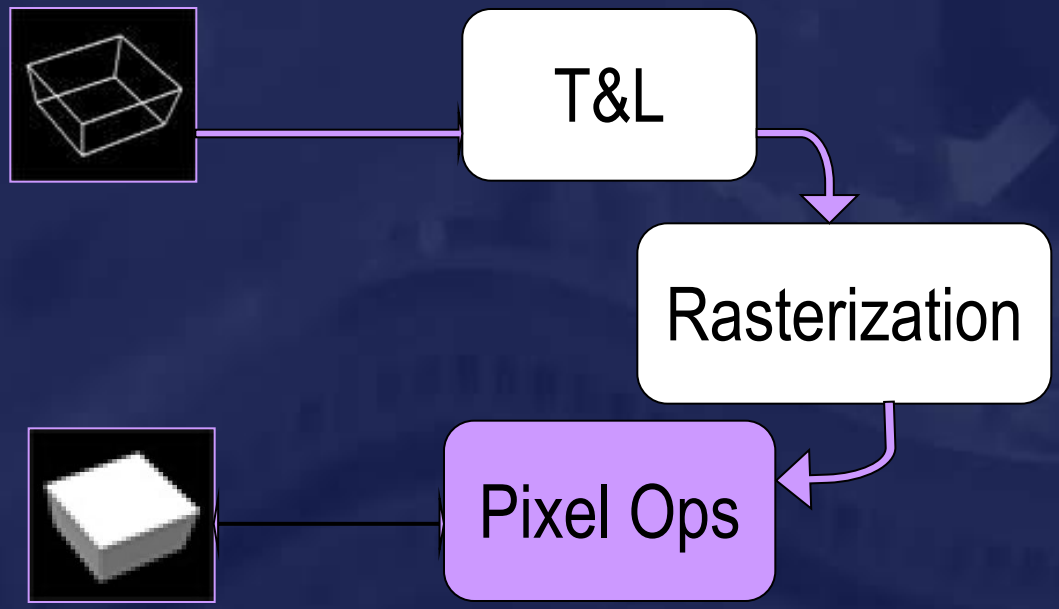
$\{ x_i,y_i,z_i,RGBA_i \}$

Pixel Ops:
Запись пиксела
в буфер кадра

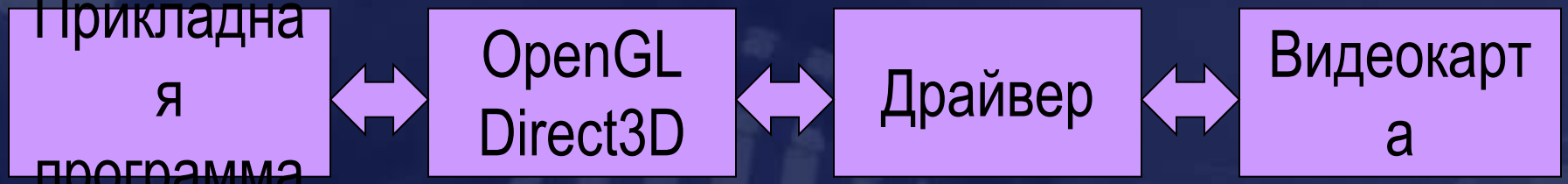
$\{ x_i,y_i,z_i,RGBA_i \}$

3D-ускорители

- “Ускоряются” этапы T&L и растеризации



- Взаимодействие с программой при помощи специальных API



Поколение 4: Шейдеры

NV25-NV47

R250-R580

```
dp4 r0.x, v0, c[0]
dp4 r0.y, v0, c[1]
dp4 r0.z, v0, c[2]
dp4 r0.w, v0, c[3]
mov oD0, c[4]  ; Output color
mov oPos, r0   ; Output vertex
```

T&L

Rasterizatio
n

Pixel
Ops

```
ps.1.0 // DX8 Version.
tex t0 // n-map.
texm3x3pad t1, t0_bx2
texm3x3pad t2, t0_bx2 v0_bx2
texm3x3tex t3, t0_bx2 dp3_sat
r0, t3_bx2,
```

OpenGL – многоплатформенная библиотека функций для создания интерактивных 2D и 3D приложений.

Отраслевой стандарт с 1992 года

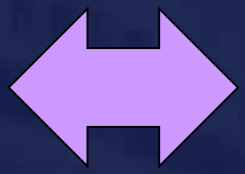
- **http://www.opengl.org**
- **http://www.opengl.org.ru**

GLut – многоплатформенная библиотека вспомогательных функций для создания оконных приложений, использующих OpenGL

Позволяет скрыть особенности программирования под данную оконную систему.

OpenGL: клиент-сервер

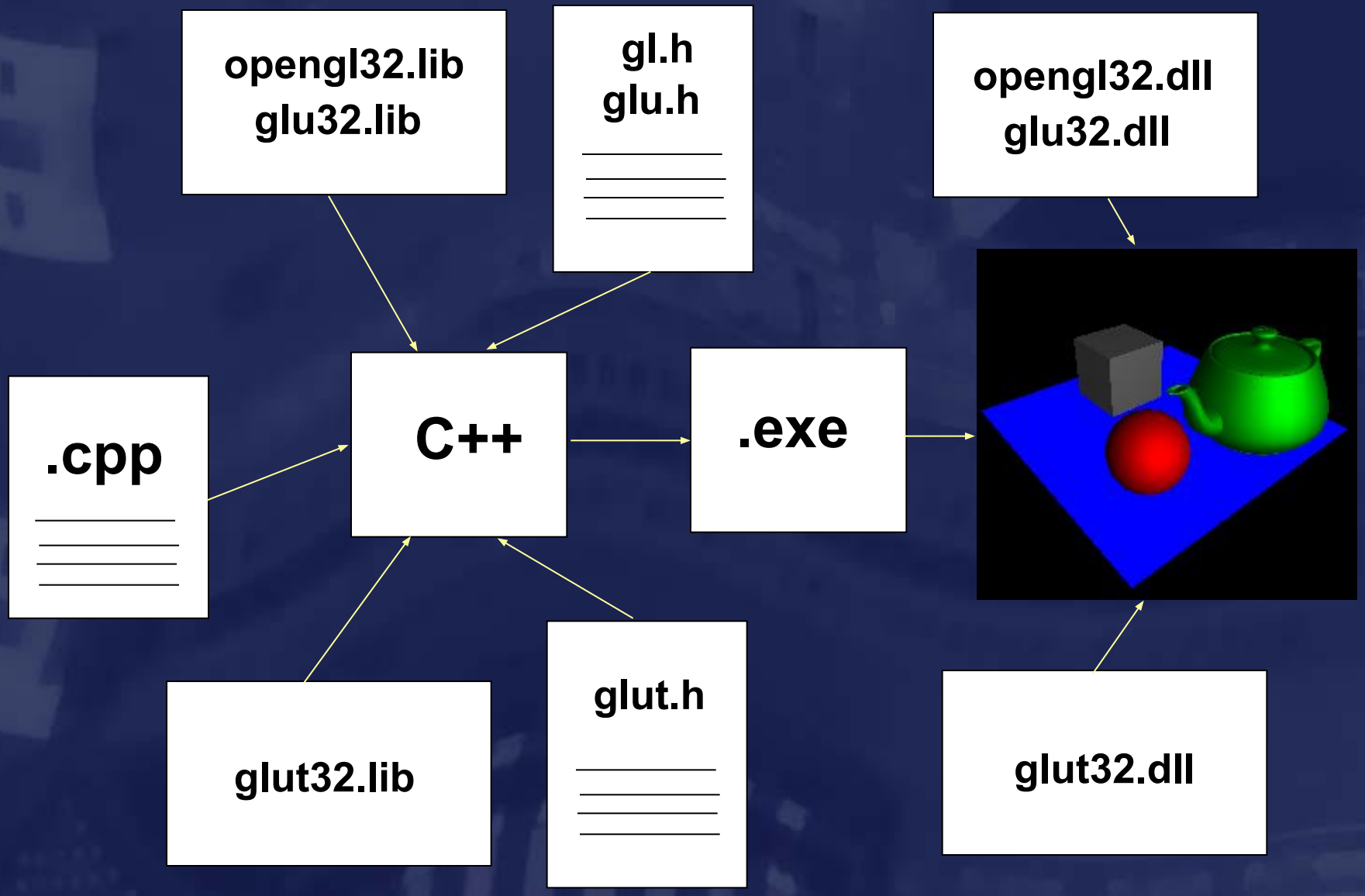
```
/* прикладная программа
*/

#include<gl/gl.h>
#include<gl/glu.h>


…
glEnable(GL_TEXTURE_2D);
glBegin(GL_TRIANGLES);
…
glTexCoord2d(0.5,0.5);
glVertex3f(1.0,0.5,-0.2);
…
glEnd();
…
```



Сервер OpenGL

Буфер кадра,
Буфер глубины,
Буфер трафарета,
Буфер аккумулятора.

Что нужно для работы с OpenGL



Литература (1/5)

Ю. Тихомиров. OpenGL. Программирование трехмерной графики, БХВ – Петербург, 2002

Эдвард Энджел. Интерактивная компьютерная графика. Вводный курс на базе OpenGL, 2-е изд., Вильямс, 2001

Литература (2/5)

Ву Мейсон, Нейдер Джеки, Девис Том, Шрайнер Дейв. OpenGL. Руководство по программиста. Диа-Софт, 2002.

Френсис Хилл. OpenGL. Программирование компьютерной графики. Для профессионалов. Питер. 2002

Литература (3/5)

Гайдуков С.OpenGL. Профессиональное программирование трехмерной графики на C++. - БХВ-Петербург, 2004

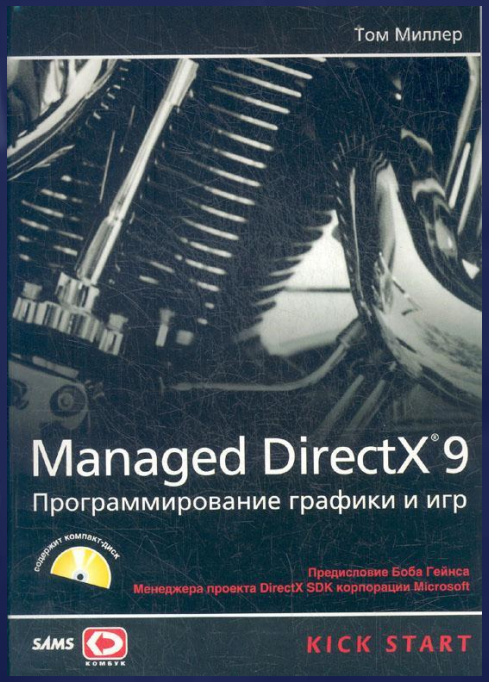
Литература (4/5)

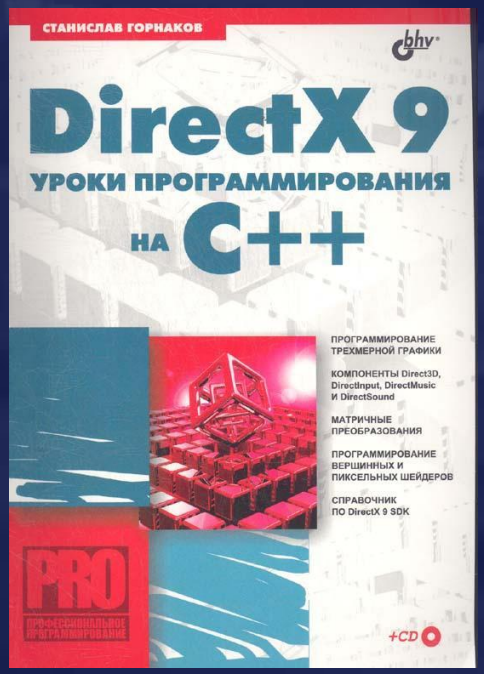
Боресков А.В. Расширения OpenGL. - СПб.: БХВ-Петербург, 2005

Дж. Рост OpenGL. Трехмерная графика и язык программирования шейдеров - СПб.: Питер, 2005

Литература (5/5)

Миллер Т. DirectX 9 с управляемым кодом. Программирование игр и графика. – КомБук, 2005.





Горнаков С. DirectX 9. Уроки программирования на C++. – БХВ, 2004.

Где взять GLut?

- http://www.opengl.org/developers/documentation/glut/index.html
- http://www.xmission.com/~nate/glut.html
- http://www.xmission.com/~nate/glut/glut-3.7.6-bin.zip
- http://www.xmission.com/~nate/glut/glut-3.7.6-src.zip

Где прочитать про GLut?

- http://www.opengl.org.ru/coding/glut/ - работа с GLut

Самая простая программа

```
#include<gl/glut.h>
#include<gl/gl.h>

void reshape(int w, int h)
  { /* Здесь обрабатываем изменение размеров окна */ }
void display(void)
  { /* Здесь помещаются команды рисования */ }
void idle(void)
  { /* Здесь происходит анимация */ }

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB);
        // GLUT_DOUBLE|GLUT_DEPTH|GLUT_STENCIL|GLUT_ACCUM
    glutCreateWindow("Самая простая программа");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(idle);
    glutMainLoop();
    return 0;
}
```

Работа с буфером кадра

- Задание цвета для заполнения буфера кадра

```
void glClearColor(GLclampf red,GLclampf green,
                  GLclampf blue,GLclampf
alpha);
```

red,green,blue,alpha $\in [0,1]$

Представление цвета в OpenGL

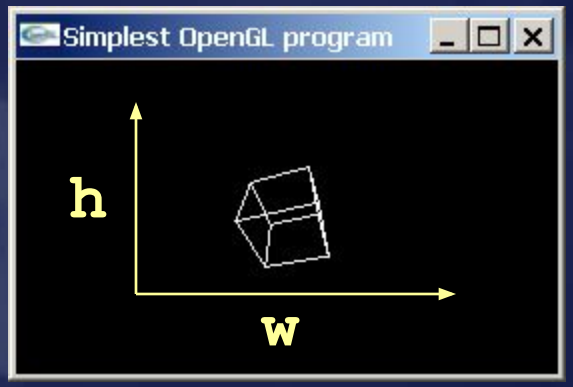
- Заполнение экранных буферов

```
void glClear(GLenum buffers);

buffers = GL_COLOR_BUFFER_BIT|
          GL_DEPTH_BUFFER_BIT|
          GL_ACCUM_BUFFER_BIT|
          GL_STENCIL_BUFFER_BIT
```

Преобразование координат: viewport

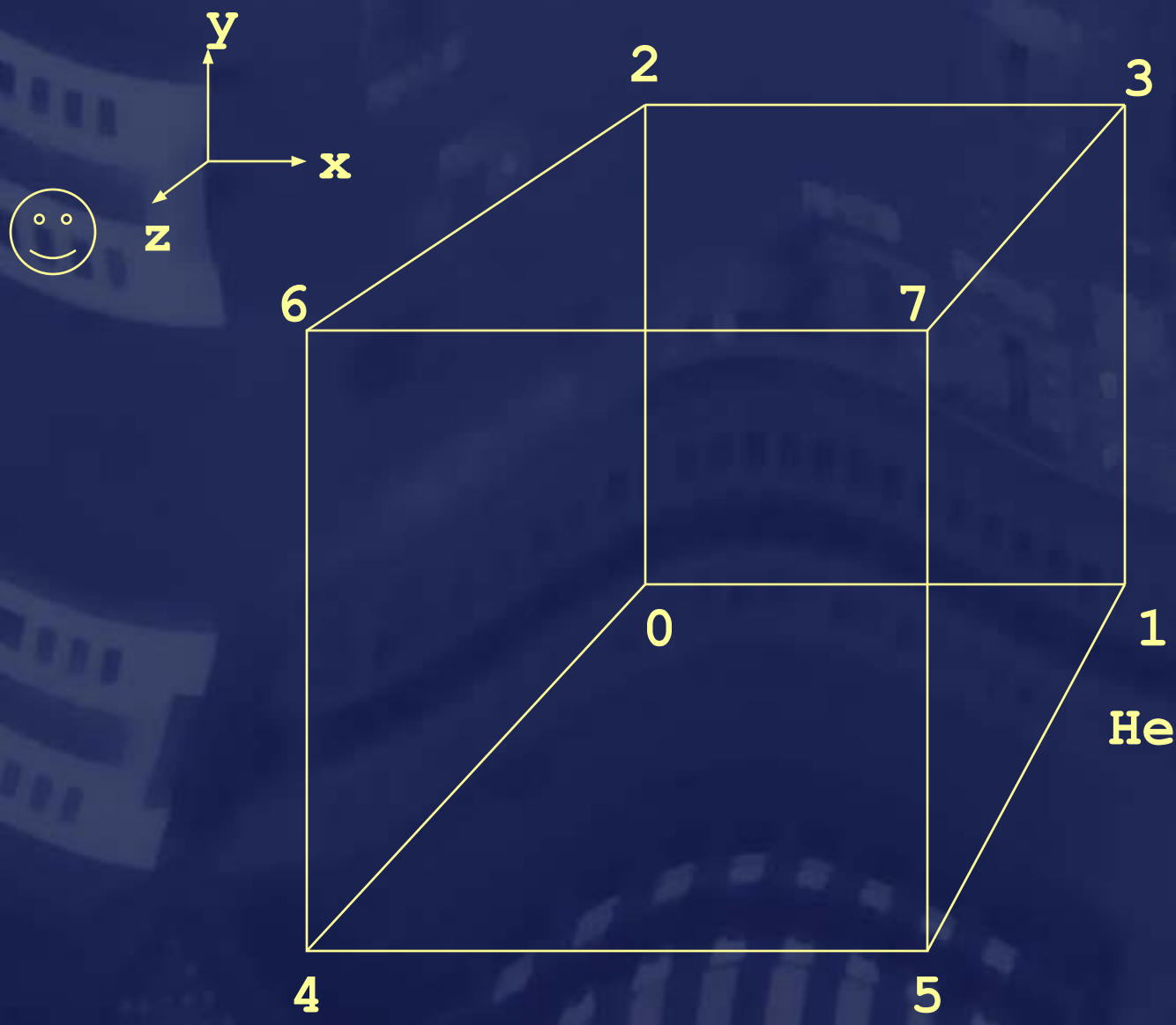
$$x_d, y_d, z_d \in [-1,1]$$
$$x_w = x + w(1 + x_d)/2$$
$$y_w = y + h(1 + y_d)/2$$
$$z_w = (f + n)/2 + z_d(f - n)/2, 0 \le n < f \le 1.$$

```
void glViewport(GLint x,GLint y,
                GLsizei w,GLsizei h);
```




```
void glDepthRange(GLclampd n,GLclampd
                  f);
```

Рисуем куб



Видимые грани:

7-6-5-4

6-7-3-2

7-5-1-3

Невидимые грани:

4-0-1-5

4-6-2-0

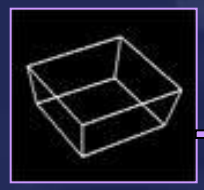
0-2-3-1

Команды OpenGL

Модель begin/end

```
void glMatrixMode(…);
void glLoadIdentity();
void glMultMatrixd(…);
```

```
void glBegin(GLenum type);
void glVertex(…);
void glNormal(…);
void glColor(…);
void glEnd();
```



T&L

Rasterization

Pixel Ops

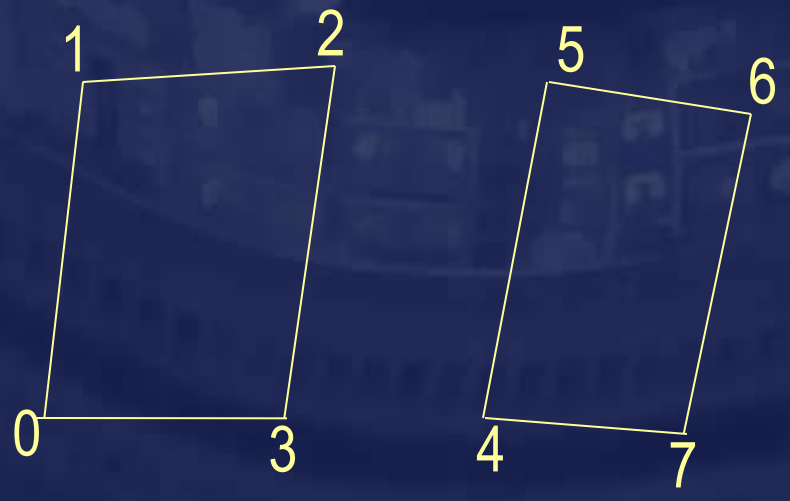
```
void glTexture2d(…);
void glTexEnv(…);
void glPolygonMode(…);
```

```
void glDepthFunc(…);
void glBlendFunc(…);
void glStencilOp(…);
```

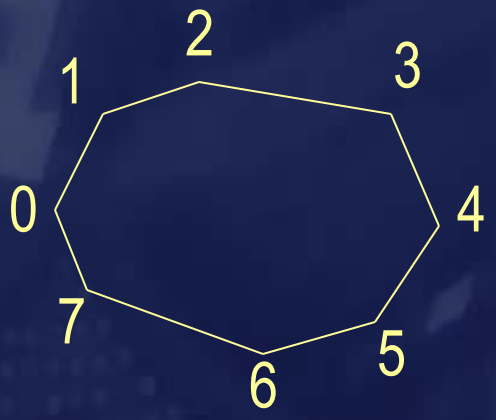
Формирование граней из вершин

- GL_TRIANGLES:

- GL_QUADS:



- GL_POLYGON:



Однородные координаты

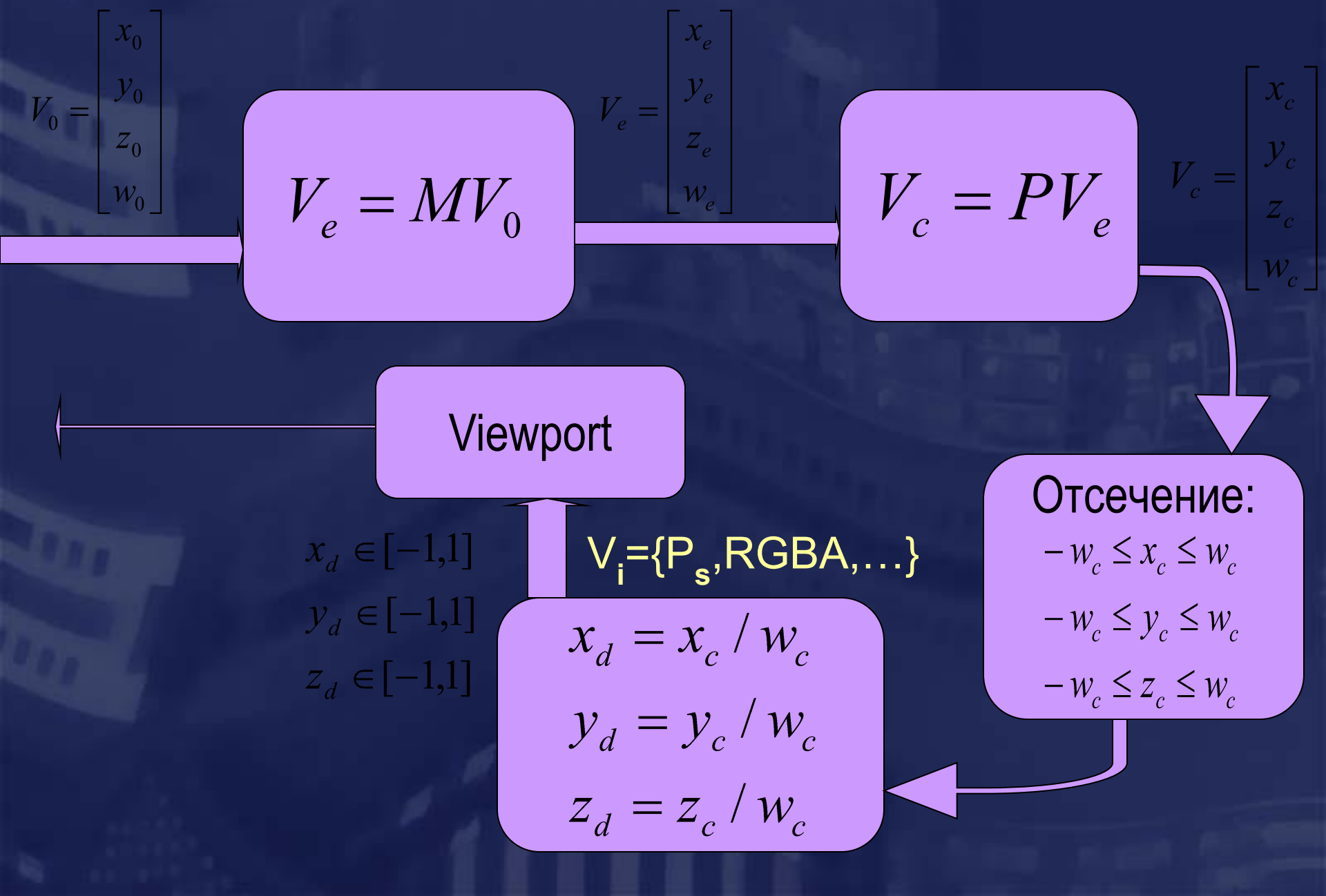
$$P = \{x, y, z, w; w \neq 0\} \longrightarrow P_3 = \{x / w, y / w, z / w\}$$

- Общее аффинное преобразование сводится к умножению на матрицу

$$T(x, y, z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Проецирование также сводится к умножению на матрицу

Преобразование координат



Матрицы преобразований

- Выбираем матрицу преобразований для изменения:

```
void glMatrixMode(Glenum mode);

mode={GL_MODELVIEW|GL_PROJECTION}
```

- Две основные операции над матрицами:

```
void glLoadIdentity();
```

$$M = E$$

```
void glMultMatrixd(GLdouble c[16]);
```

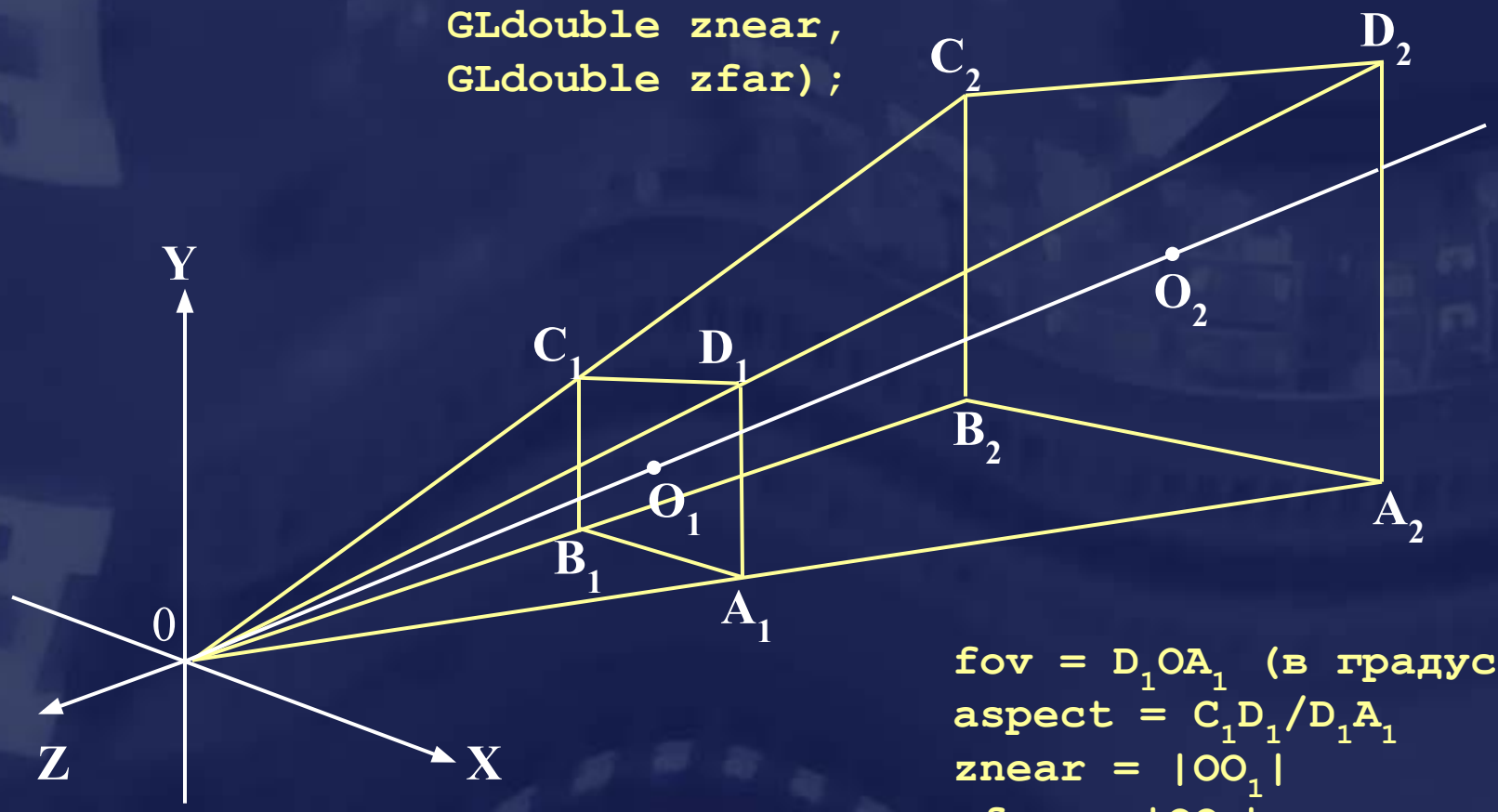
$$M = M \cdot \begin{bmatrix} c[0] & c[4] & c[8] & c[12] \\ c[1] & c[5] & c[9] & c[13] \\ c[2] & c[6] & c[10] & c[14] \\ c[3] & c[7] & c[11] & c[15] \end{bmatrix}$$

Матрицы преобразований. Продолжение

```
void glTranslated(GLdouble x,
                  GLdouble y,
                  GLdouble z);

void glScaled(GLdouble x,
              GLdouble y,
              GLdouble z);

void glRotated(GLdouble angle,
               GLdouble ax,
               GLdouble ay,
               GLdouble az);

void gluPerspective(GLdouble fov,
                    GLdouble aspect,
                    GLdouble znear,
                    GLdouble zfar);
```

Как работает gluPerspective?

```
void gluPerspective(GLdouble fov,
                    GLdouble aspect,
                    GLdouble znear,
                    GLdouble zfar);
```



fov = D_1OA_1 (в градусах)

aspect = C_1D_1/D_1A_1

znear = $|OO_1|$

zfar = $|OO_2|$

gluPerspective: продолжение

$$P = \begin{bmatrix} \frac{f}{a} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_f + z_n}{z_n - z_f} & \frac{2 z_f z_n}{z_n - z_f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$f = ctg(\texttt{fovy}/2),$

$a = \texttt{aspect}$

$z_n = \texttt{znear}$

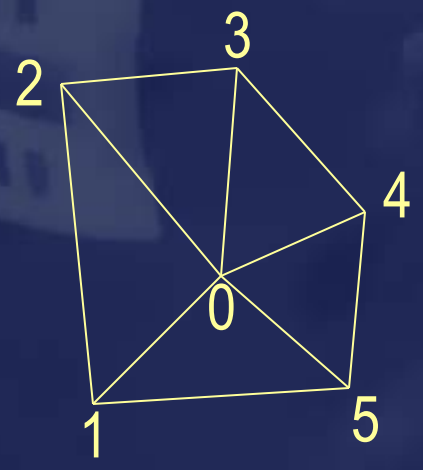
$z_f = \texttt{zfar}$

$$O_1 : P \begin{bmatrix} 0 \\ 0 \\ -z_n \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ z_n \\ -z_n \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

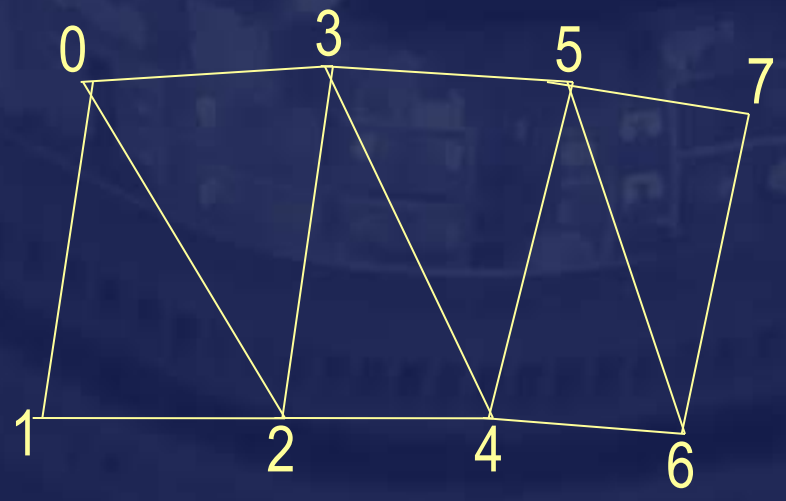
$$O_2 : P \begin{bmatrix} 0 \\ 0 \\ -z_f \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -z_f \\ -z_f \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Уменьшение количества вершин

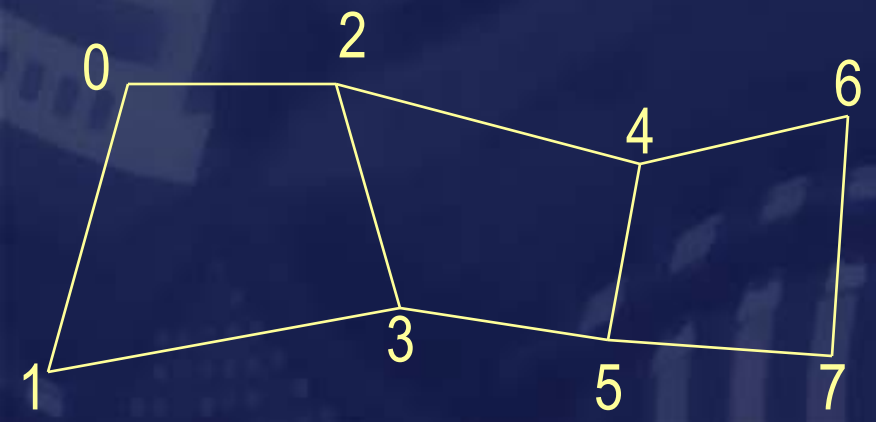
- GL_TRIANGLE_FAN: 3n vs. 1+n, n>1



- GL_TRIANGLE_STRIP: 3n vs. 2+n



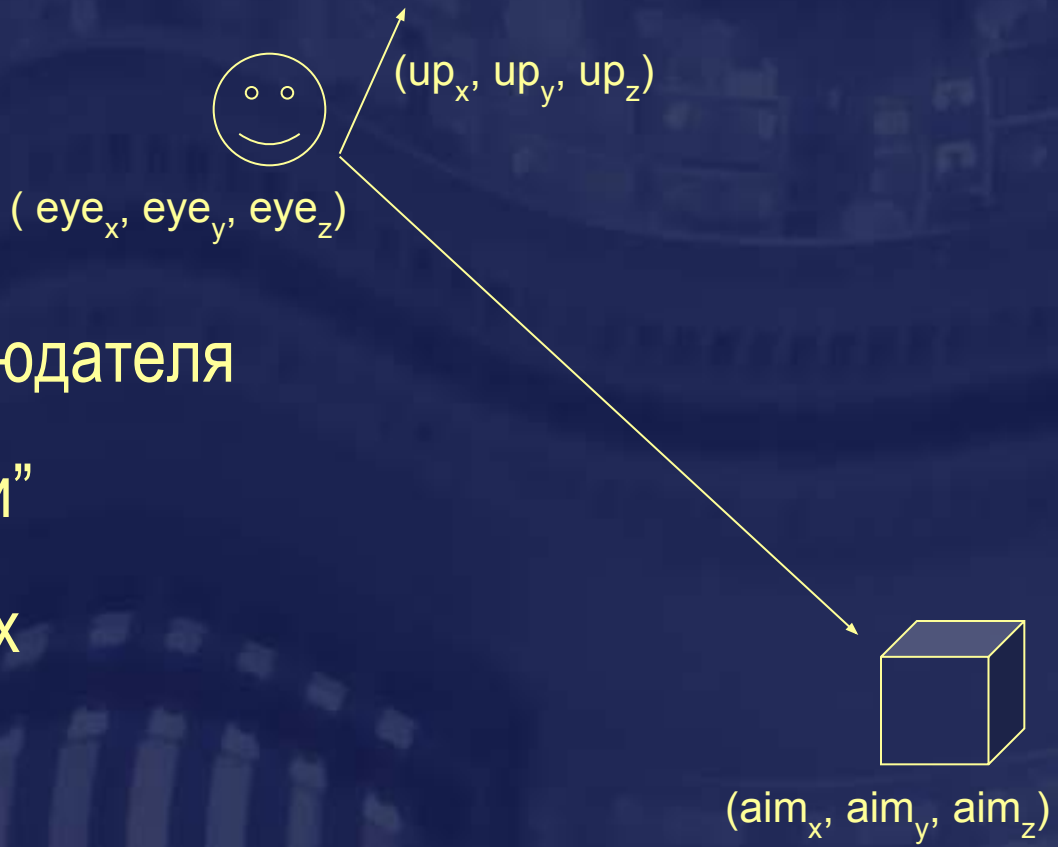
- GL_QUAD_STRIP: 4n vs. 2+2n



Виртуальная камера

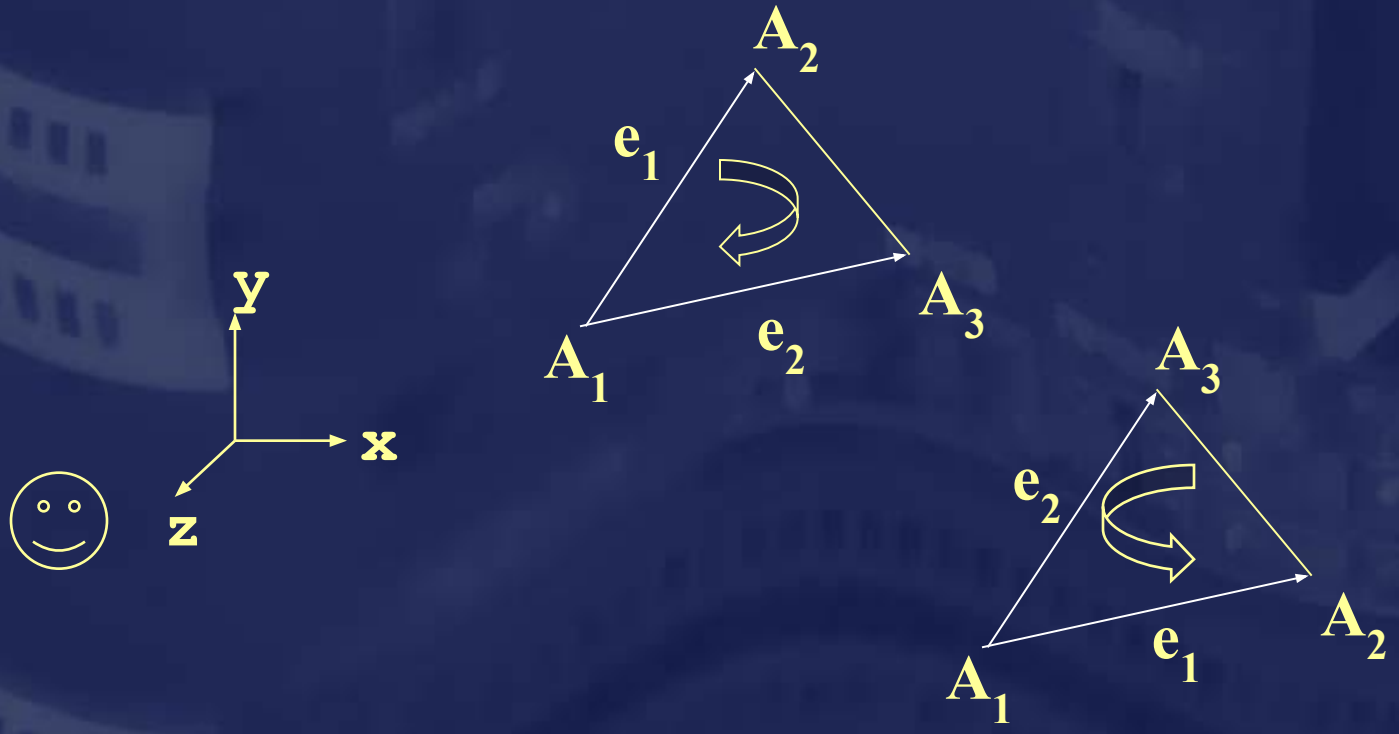
- Настройка виртуальной камеры

gluLookAt(eye_x, eye_y, eye_z, aim_x, aim_y, aim_z, up_x, up_y, up_z)



- eye – координаты наблюдателя
- aim – координаты “цели”
- up – направление вверх

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)}
```

Дисплейные списки

- Дисплейный список (display list) – запомненная последовательность команд OpenGL.
- Находим неиспользуемый номер дисплейного списка

```
GLuint n = glGenLists(1);
```

- Сохраняем последовательность команд

```
glNewList(n,GL_COMPILE);
<…вызовы функции OpenGL…>
glEndList();
```

- Воспроизводим сохраненную последовательность команд (с теми же самыми параметрами!)

```
glCallList(n);
```

- Освобождаем номер дисплейного списка

```
glDeleteLists(n,1);
```

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

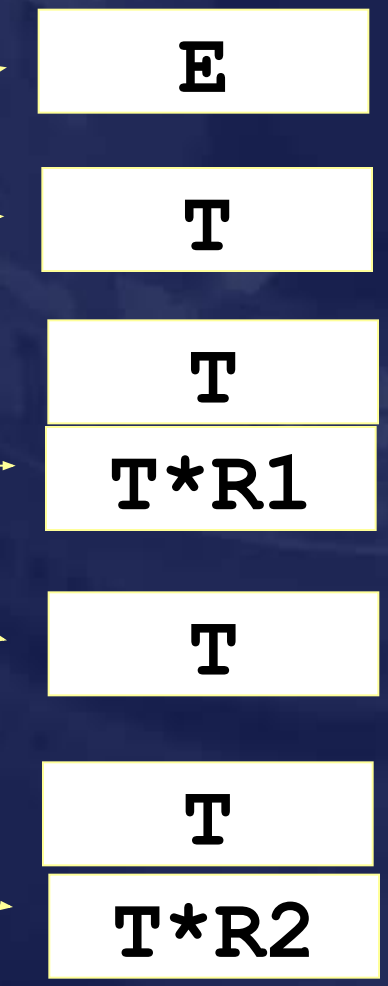
```
glDepthMask(TRUE); или glDepthMask(FALSE);
```

- Возможно задать операцию сравнения

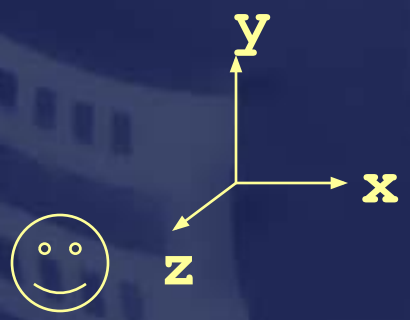
```
glDepthFunc(GLenum type);
type = {GL_ALWAYS|GL_NEVER|GL_LESS|GL_GREATER|
        GL_EQUAL|GL_NOTEQUAL|GL_LEQUAL|GL_GEQUAL}
```

Стек матриц

```
glLoadIdentity();
glTranslated(…);

glPushMatrix();
glRotated(…);
glPopMatrix();

glPushMatrix();
glRotataed(…);
glPopMatrix();
```



Рисуем тор

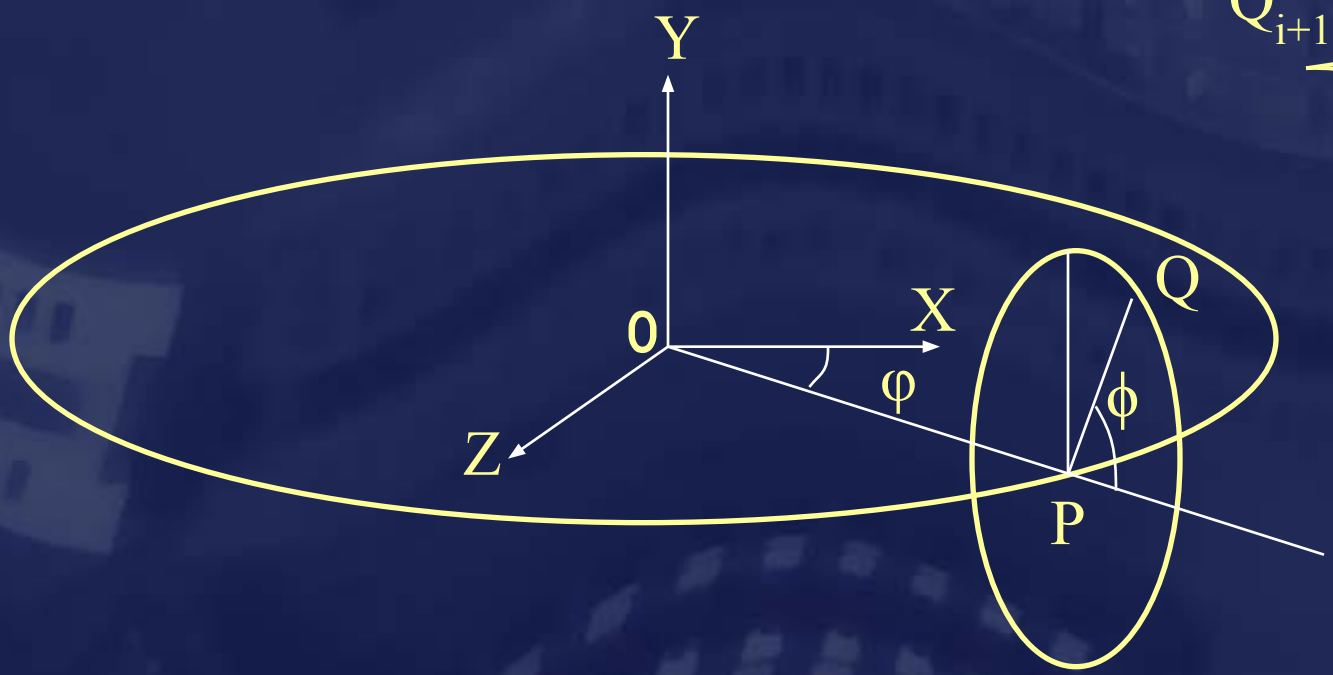


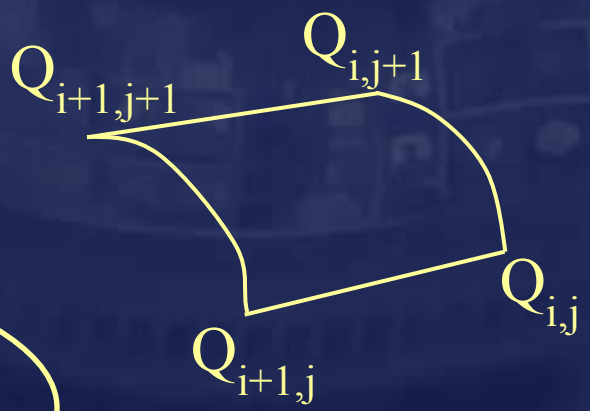
$$P(\phi) = (R_1 \cos\phi, 0, R_1 \sin\phi)$$

$$\bar{n}(\phi,\varphi) = (\cos\phi\cos\varphi, \sin\varphi, \sin\phi\cos\varphi)$$

$$Q(\phi,\varphi) = P(\phi) + R_2\bar{n}(\phi,\varphi)$$

$$\phi,\varphi \in [0,2\pi]$$



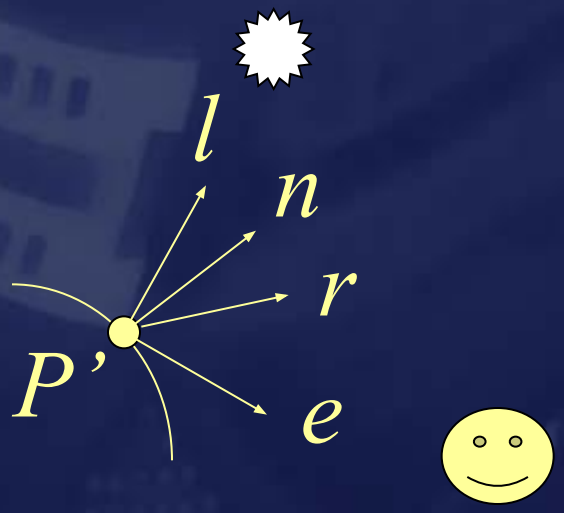


$$Q_{i,j} = Q(2\pi i / N_1, 2\pi j / N_2),\ i = 0,1,\ldots,N_1,\ j = 0,1,\ldots,N_2.$$

Уравнение освещенности по Фонгу

$$I = a_m a_l + d_m d_l (n \boxtimes l) + s_m s_l (e \boxtimes r)^{h_s}$$

- Фоновое освещение не имеет источника и зависит только от сцены
- При диффузном освещении свет от источника равномерно рассеивается во всех направлениях.
- При зеркальном освещении свет от источника отражается от повехности.в одном направлении. Зеркальная освещенность дополнительно зависит от положения наблюдателя..



$$(a \boxtimes b) = \begin{cases} (a,b), & (a,b) \geq 0 \\ 0, & (a,b) < 0 \end{cases}$$

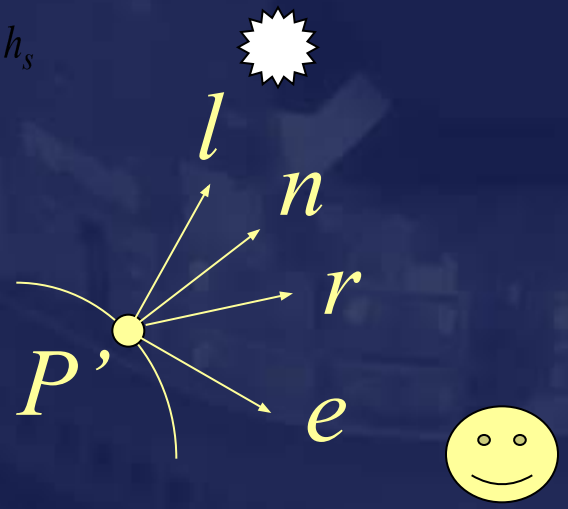
$$r = reflect(l,n)$$

Модели Блинна и Шлика

- Вычисление отраженного вектора – трудоемкая операция (Блинн)

$$I = a_m a_l + d_m d_l (n \cdot l) + s_m s_l (n \cdot h)^{h_s}$$

$$h = \frac{l + e}{\|l + e\|}$$



- Возведение в степень также работает не очень быстро... (Шлик)

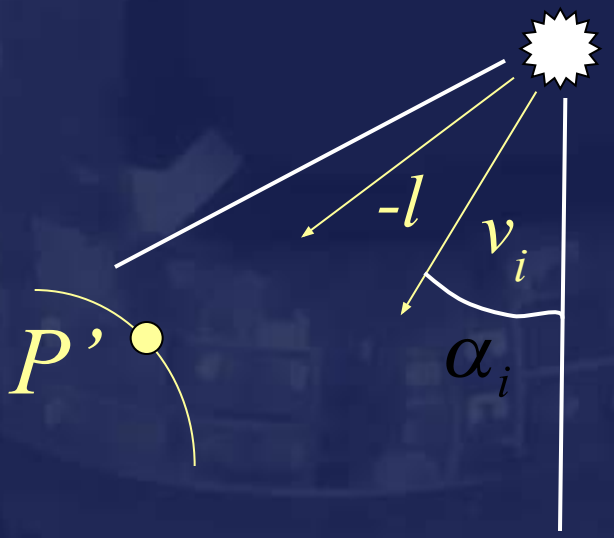
$$(n \cdot h)^{h_s} \sim \frac{D}{h_s - D h_s + D}, D = (n \cdot h)$$

Уравнение освещенности OpenGL

$$c = e_m + a_m a_s + \sum_{i=0}^{n-1} att_i \cdot spot_i \cdot (a_m a_i + d_m d_i (n \cdot l) + s_m s_i (n \cdot h)^{h_m})$$

$$att_i = \frac{1}{k_{c,i} + k_{l,i} r + k_{q,i} r^2},$$

$$spot_i = \begin{cases} 1, \alpha_i = \pi, \\ 0, (v_i, -l) < \cos(\alpha_i), \\ (v_i, -l), (v_i, -l) \geq \cos(\alpha_i). \end{cases}$$



$spot_i$ – коэффициент направленности

att_i – коэффициент затухания

a_s – фоновое освещение

a_i, s_i, d_i – свойства i-го источника освещения

e_m, a_m, s_m, d_m, h_m – свойства материала

Установка параметров освещения в OpenGL

- Задаем параметры материала:

```
void glMaterialfv(GLenum face,GLenum param,GLfloat *value);
    face = {GL_FRONT|GL_BACK}
    param = {GL_AMBIENT|GL_DIFFUSE|GL_EMISSIVE|GL_SPECULAR}
    value = float[4] // RGBA

void glMaterialf(GLenum face,GL_SHININESS,GLfloat value);
```

- Задаем цвет фонового освещения:

```
void glLightModelfv(GLenum param,GLfloat *value);
    param = LIGHT_MODEL_AMBIENT
    value = float[4] // RGBA
```

- Задаем цвет источника освещения:

```
void glLightfv(GLenum light,GLenum param,GLfloat *value);
    face = {GL_LIGHT0|GL_LIGHT1|…}
    param = {GL_AMBIENT|GL_DIFFUSE|GL_SPECULAR}
    value = float[4] // RGBA
```

Установка параметров освещения. Часть 2.

- Задаем положение источника освещения:

```
void glLightfv(GLenum light, GL_POSITION,GLfloat *value);
    face = {GL_LIGHT0|GL_LIGHT1|…}
    value = float[4] // x,y,z,w
```

Координаты источника освещения преобразуются текущей матрицей модельного преобразования!

- Включаем расчет освещенности

```
void glEnable(GLenum type); type = GL_LIGHTING;
```

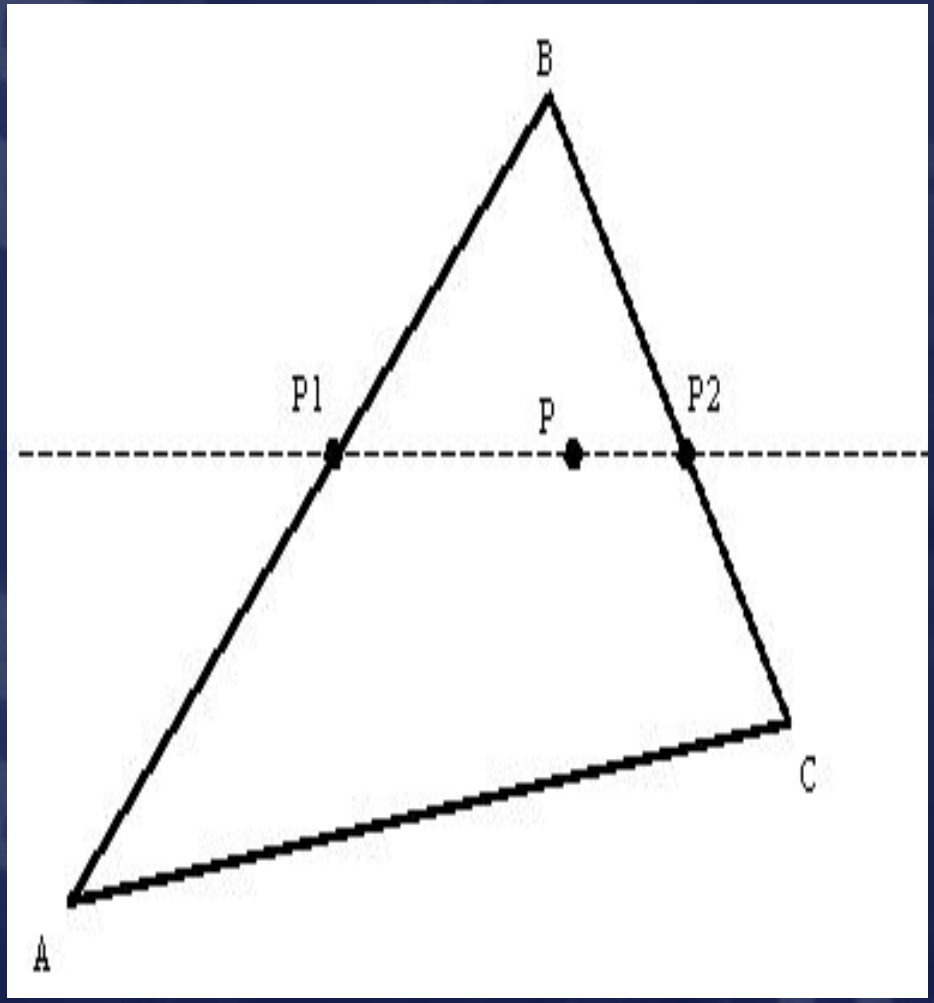
- Включаем требуемые источники освещения

```
void glEnable(GLenum type); type = GL_LIGHT0;
```

- Включаем требуемые источники освещения

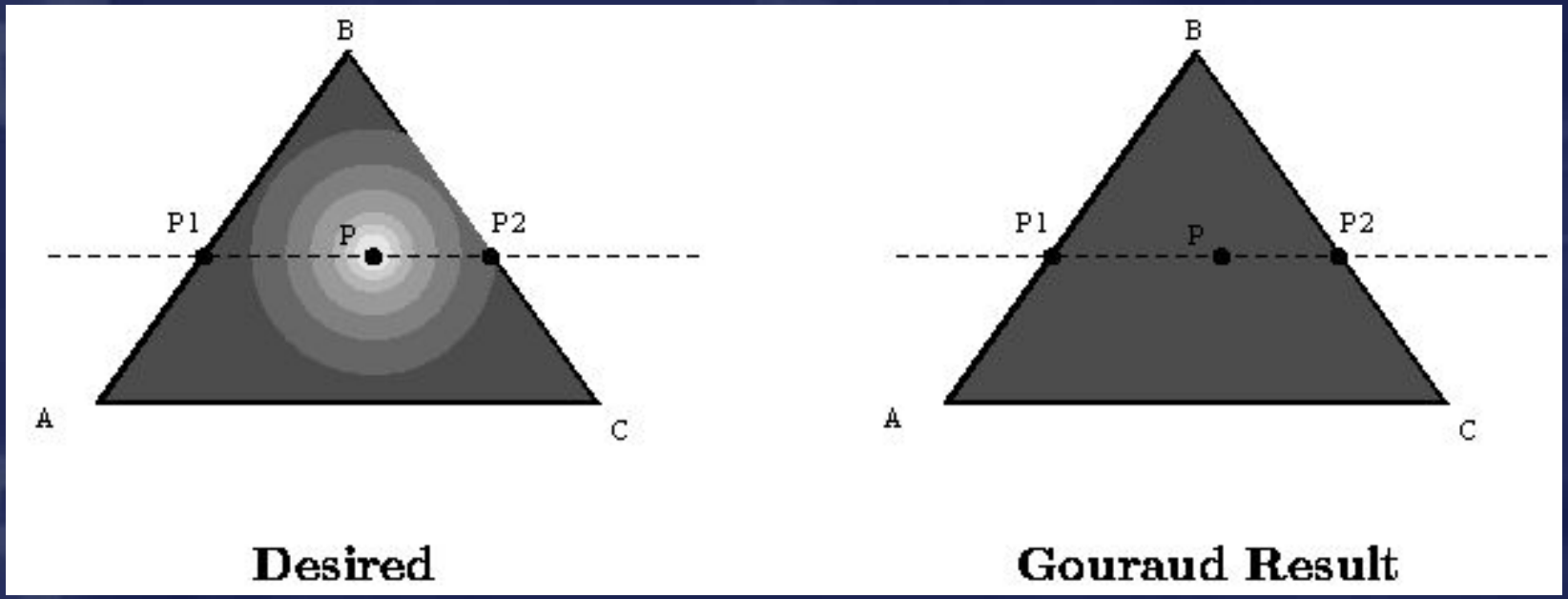
```
void glShadeModel(GLenum type);
    type = GL_FLAT; - плоская закраска грани
    type = GL_SMOOTH - закраска по Гуро
```

Интерполяция цвета

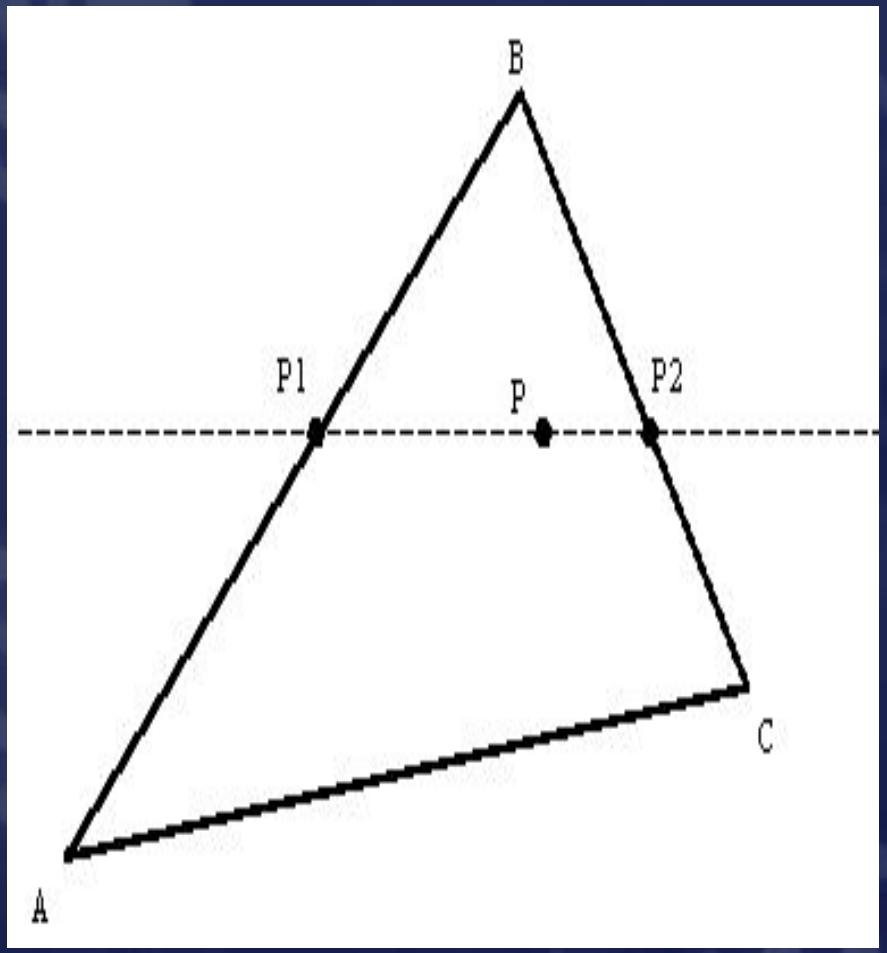


- Вычислить цвет (RGB) в каждой вершине.
- Вычислить цвет в точках P1 и P2:
  $s = ||P1 - B|| / ||A - B||$
  $C(P1) = s(C(A)) - (1-s)(C(B))$
- Вычислить цвет в т. P:
  $s = ||P - P2|| / ||P1 - P2||$
  $C(P) = s(C(P1))-(1-s)(C(P2))$

Недостатки закраски по Гуро



Интерполяция нормали



- Вычислить нормали (RGB) в каждой вершине.
- Вычислить нормаль в точках P1 и P2:
  $s = ||P1 - B|| / ||A - B||$
  $N(P1) = s(N(A)) + (1-s)(N(B))$
- Вычислить нормаль в т. P:
  $s = ||P - P2|| / ||P1 - P2||$
  $N(P) = s(т(P1))-(1-s)(N(P2))$
- Вычислить цвет в точке P.

Массивы вершин

- Задаем массив вершин, нормалей и текстурных координат:

```
void glVertexPointer(GLint size,GLenum type,
        GLsizei stride,const GLvoid *pointer;
```

1	1	-1
1	-1	-1
-1	-1	-1
-1	1	-1
1	1	1
1	-1	1
-1	-1	1
-1	1	1

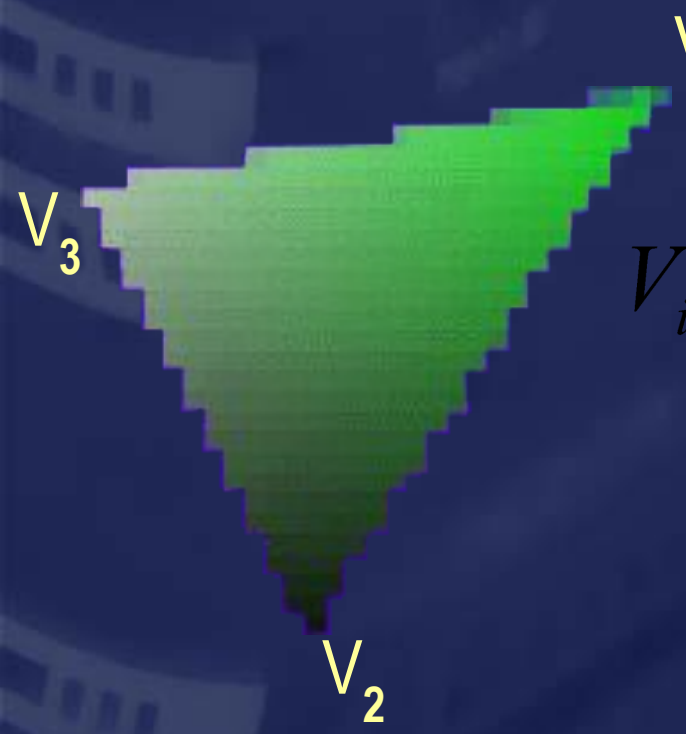
2	3	7	6	...

- Задаем последовательность номеров вершин

```
void glDrawElements(GLenum mode,GLsizei count,
  GLenum type, const GLvoid *indices);
```

- Возможно полностью избежать дублирования вершин

Растеризация



$$V_i = \{x_i, y_i, z_i, RGBA_i, ...\}, i = 1,2,3$$

- Интерполяция координаты z

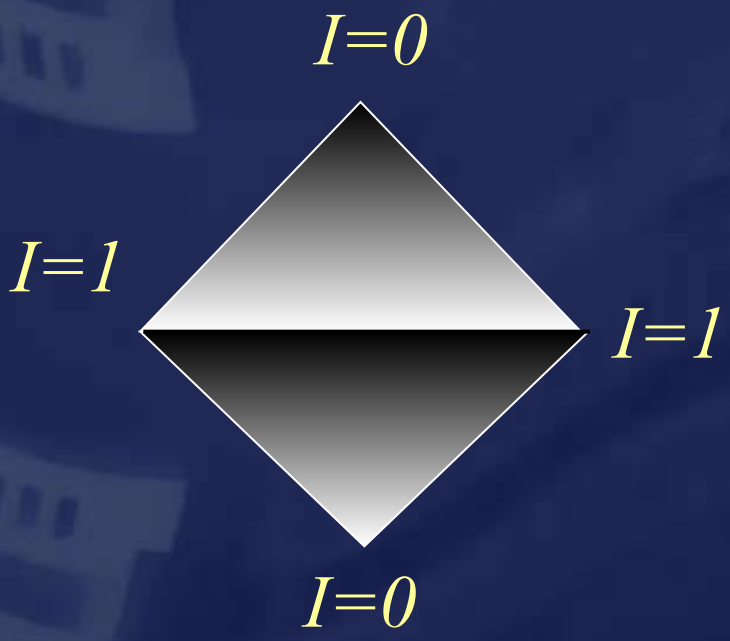
$$z(x, y) = L(x, y, z_i)$$

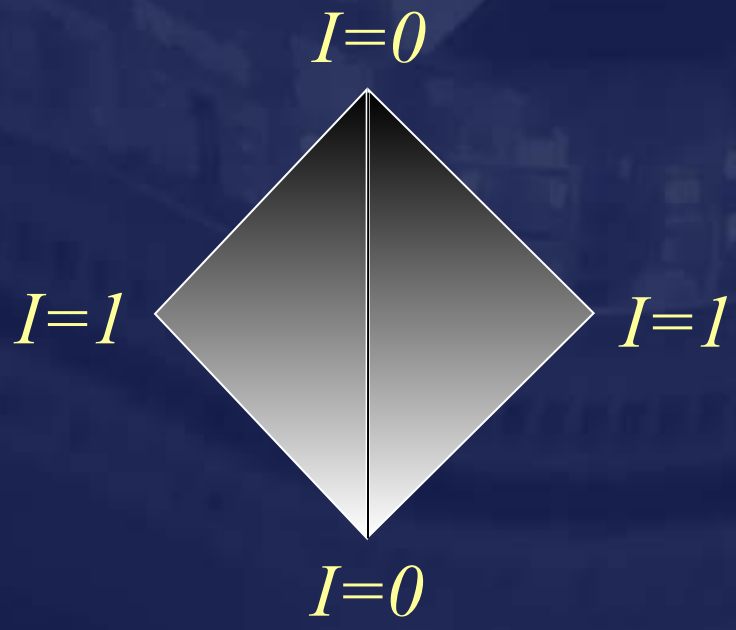
- Интерполяция цвета вдоль примитива - закраска по Гуро

$$RGBA(x, y) = L(x, y, RGBA_i)$$

Ошибки линейной интерполяции

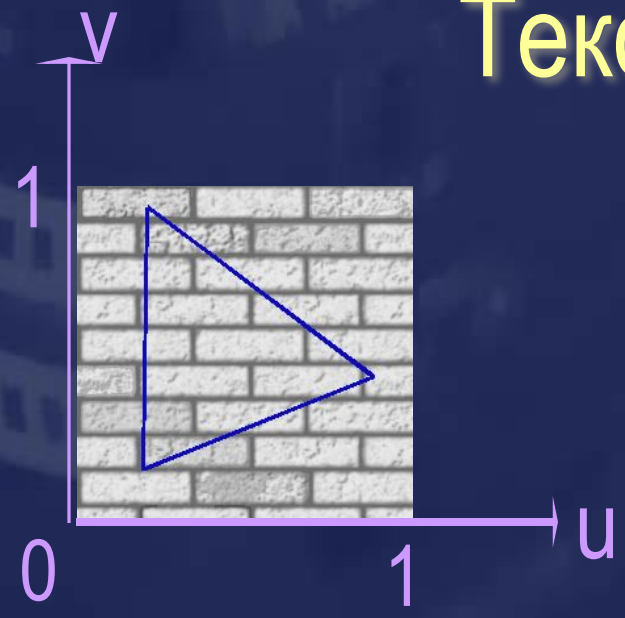
- Освещенность зависит от способа разбиения на примитивы





- Поле нормалей лучше задавать в виде текстуры!

Текстурирование



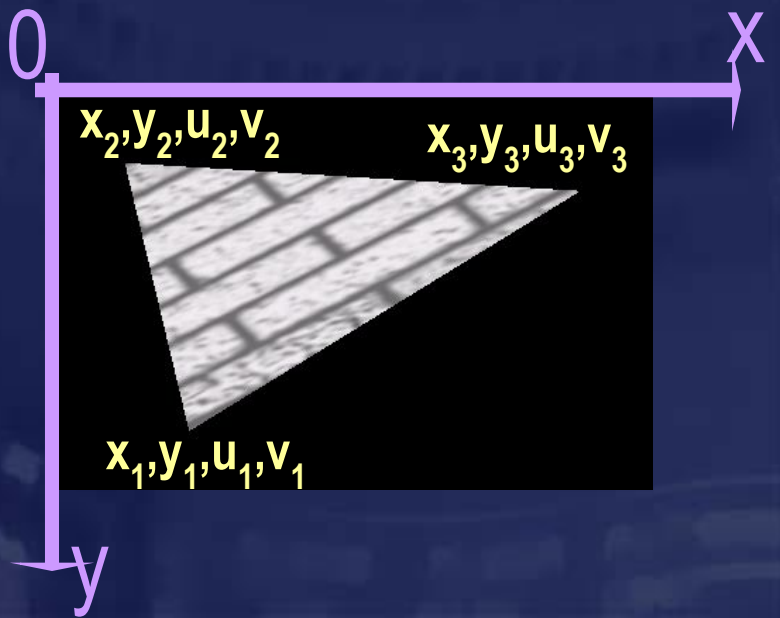
$$V_i = \{P_i, n_i, u_i, v_i, \ldots\}, i = 1,2,3$$

$$V_i' = \{x_i, y_i, u_i, v_i, \ldots\}, i = 1,2,3$$

“Перспективное” текстурирование:

$$u(x,y) = \frac{Ax + By + C}{Px + Qy + R}$$

$$v(x,y) = \frac{Dx + Ey + F}{Px + Qy + R}$$



Текстурирование в OpenGL

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

RGB_{00}	RGB_{10}	...	RGB_{N0}
RGB_{01}	RGB_{11}	...	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: часть 2

- Загружаем текстуру из памяти в текстурный объект:

```
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: часть 3

- Разрешаем текстурирования

```
glEnable(GL_TEXTURE_2D);
```

- Задаем текстурные координаты (обычно для каждой вершины)

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

- Возможно, потребуется включить режим перспективного текстурирования

```
glHint(GL_PERSPECTIVE_CORRECTION_HINT,GL_NICEST);
```

- Возвращаем номер текстурного объекта в список свободных

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

Как загрузить картинку из файла?

- Функции BmpLoad из примеров:

```
unsigned char *LoadIndexedBMPFile
    (const char *path,int *width,int *height);
unsigned char *LoadTrueColorBMPFile
    (const char *path,int *width,int *height);
```

- Классы из NV SDK:

```
namespace jpeg
{
    extern int read(const char *filename,
     int *width,int *height,
     unsigned char **pixels, int *components);
}
```

- Воспользоваться любой другой сторонней библиотекой

Фильтрация текстур

$$s = 2^n u, \quad t = 2^m v$$

- Выборка ближайшего текселя GL_NEAREST:

$$i = \lfloor s \rfloor, j = \lfloor t \rfloor$$

$$\tau = \tau_{i,j}$$

- Линейная комбинация 4-х соседних пикселей GL_LINEAR:

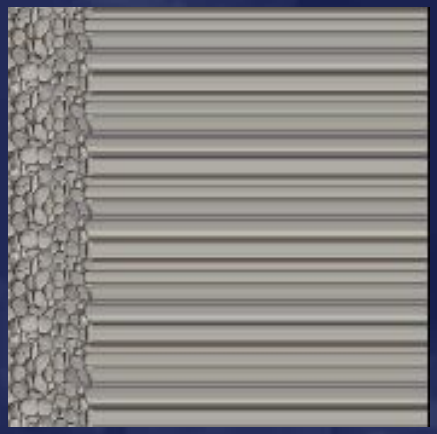
$$\alpha = frac(s - 1/2), \quad \beta = frac(t - 1/2)$$

$$i_0 = \lfloor s \rfloor, j_0 = \lfloor t \rfloor, \quad i_1 = i_0 + 1, j_1 = j_0 + 1$$

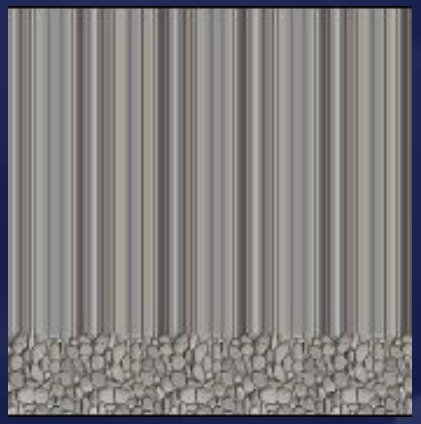
$$\tau = (1-\alpha)(1-\beta)\tau_{00} + \alpha(1-\beta)\tau_{10} + (1-\alpha)\beta\tau_{01} + \alpha\beta\tau_{11}$$

Свертка текстурных координат

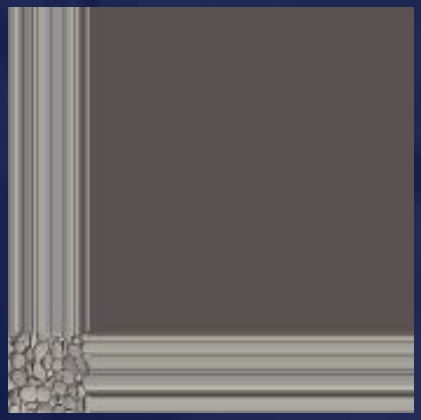
```
glTexParameteri(…,GL_REPEAT);
glTexParameteri(…,GL_REPEAT);
```



```
glTexParameteri(…,GL_CLAMP);
glTexParameteri(…,GL_REPEAT);
```



```
glTexParameteri(…,GL_REPEAT);
glTexParameteri(…,GL_CLAMP);
```



```
glTexParameteri(…,GL_CLAMP);
glTexParameteri(…,GL_CLAMP);
```

Фильтрация текстур: mipmapping

$$\rho = \max\left\{\sqrt{\left(\frac{\partial s}{\partial x}\right)^2 + \left(\frac{\partial t}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial s}{\partial y}\right)^2 + \left(\frac{\partial t}{\partial y}\right)^2}\right\}$$

$0 < \rho < 1$ увеличение текстуры

$\rho = 1$ масштаб 1:1

$\rho > 1$ уменьшение текстуры

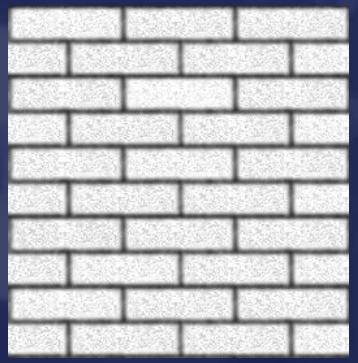
$$\lambda = \log_2 \rho$$

$\lambda < 0$ увеличение текстуры

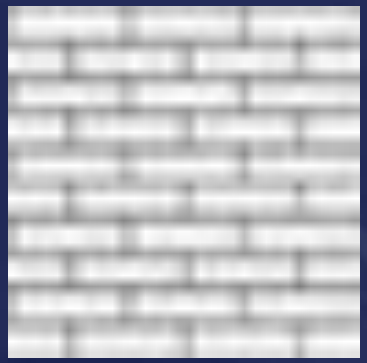
$\lambda = 0$ масштаб 1:1

$\lambda > 0$ уменьшение текстуры в 2^{λ} раз

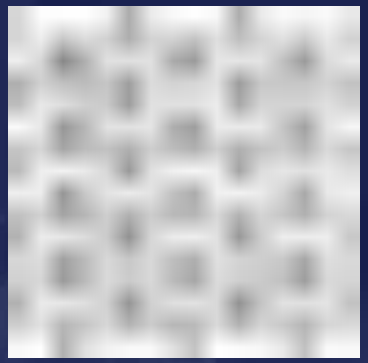
Фильтрация текстур: mipmapping. Часть 2



256x256, $\lambda=0$



64x64, $\lambda=2$



16x16, $\lambda=4$

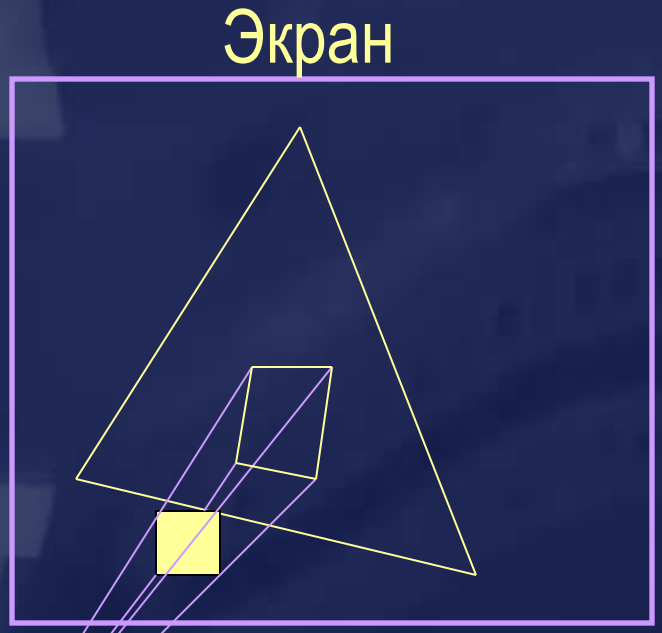
- Трилинейная фильтрация GL_LINEAR_MIPMAP_LINEAR

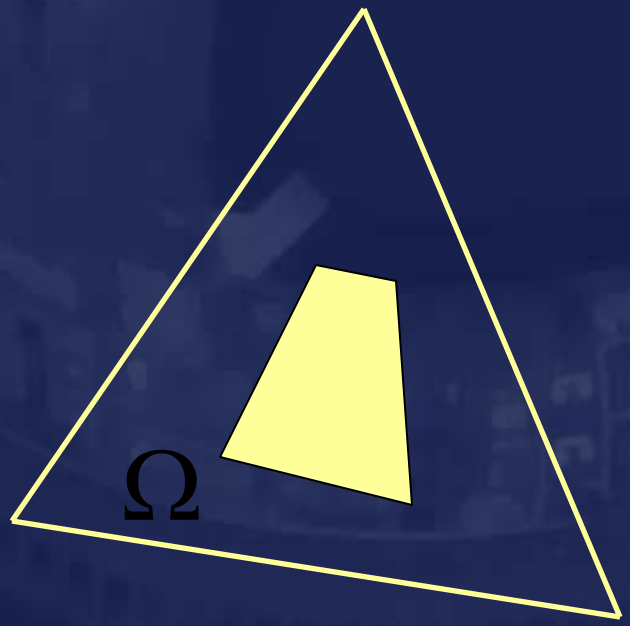
$$0 \le d-1 \le \lambda < d, d - \text{целое}$$

$$\alpha = frac(\lambda)$$

$$\tau = \alpha\tau_{d-1} + (1-\alpha)\tau_d$$

Анизотропная фильтрация

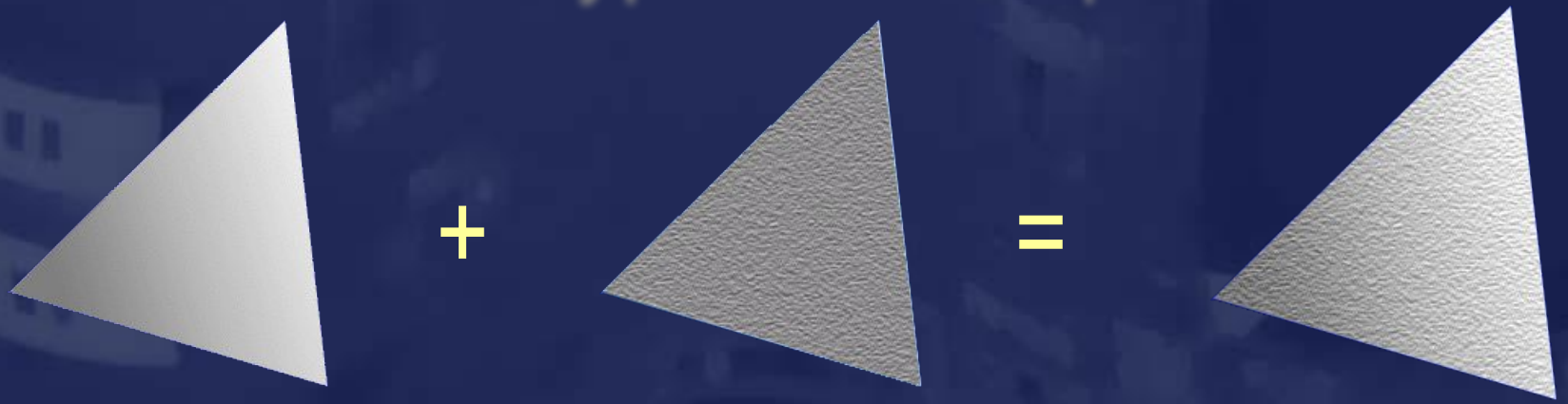




$$\tau = \sum_{i=0}^{15} \alpha_i \tau_{x_i y_i}, (x_i, y_i) \in \Omega$$

Расширение "GL_EXT_texture_filter_anisotropic"

Текстура и освещение



$RGBA_f$ из модуля T&L $RGBA_t$ текстуры $RGBA_c$ результат

- Замещение GL_REPLACE

$$R_c = R_t, G_c = G_t, B_c = B_t, A_c = A_t$$

Demo

- Модулирование GL_MODULATE (по умолчанию)

$$R_c = R_f R_t, G_c = G_f G_t, B_c = B_f B_t, A_c = A_f A_t$$

- Смешение GL_DECAL,GL_BLEND

$$R_c = R_f (1 - A_t) + R_t A_t, A_c = A_f$$

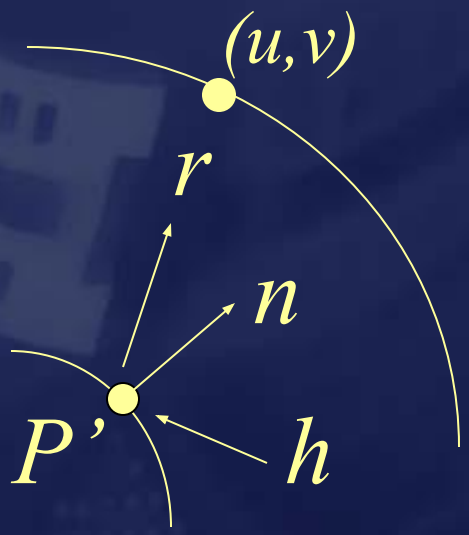
Генерация текстурных координат

- Линейная зависимость

$$u = a_{11}P_x + a_{12}P_y + a_{13}P_z + a_{14}P_w$$

$$v = a_{21}P_x + a_{22}P_y + a_{23}P_z + a_{24}P_w$$

- Environment mapping - эффект отражающей поверхности



$$r = u - 2(n, n^T)h$$

$$m = 2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}$$

$$u = 1/2 + r_x / m$$

$$v = 1/2 + r_y / m$$

Генерация текстурных координат. Продолжение

- Включаем автоматическую генерацию текстурных координат (для первыхдвух координат)

```
glEnable(GL_TEX_GEN_S);
glEnable(GL_TEX_GEN_T);
```

- Включаем автоматическую генерацию текстурных координат (для первых двух координат)

```
glTexGeni(GL_S,GL_TEXTURE_GEN_MODE,GL_SPHERE_MAP);
glTexGeni(GL_T,GL_TEXTURE_GEN_MODE,GL_SPHERE_MAP);
```

Использование текстуры как фонового изображения

- Устанавливаем ортогональную проекцию

```
glMatrixMode(GL_PROJECTION);
glLoadIndentity();
glOrtho(-1,1,-1,1,-1,1);
```

- Устанавливаем ортогональную проекцию

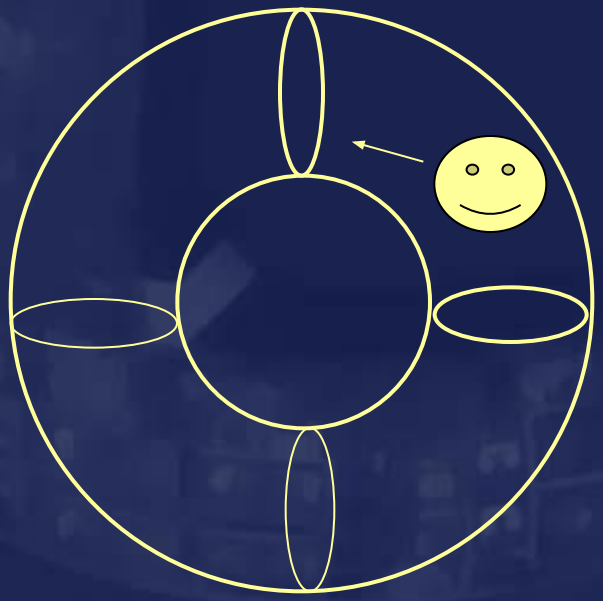
```
glMatrixMode(GL_MODELVIEW);
glLoadIndentity();
```

- Рисуем прямоугольник

```
glEnable(GL_TEXTURE_2D);
glBindTexture(bktex);
glBegin(GL_QUADS);
glTexCoord2f(0,0); glVertex2f(-1,-1);
glTexCoord2f(1,0); glVertex2f(1,-1);
glTexCoord2f(1,1); glVertex2f(1,1);
glTexCoord2f(0,1); glVertex2f(-1,1);
glEnd();
```

Преобразование текстурных координат

$$\begin{bmatrix} u' \\ v' \\ p' \\ q' \end{bmatrix} = T \begin{bmatrix} u \\ v \\ p \\ q \end{bmatrix} \rightarrow \begin{bmatrix} u'/q' \\ v'/q' \\ p'/q' \end{bmatrix}$$

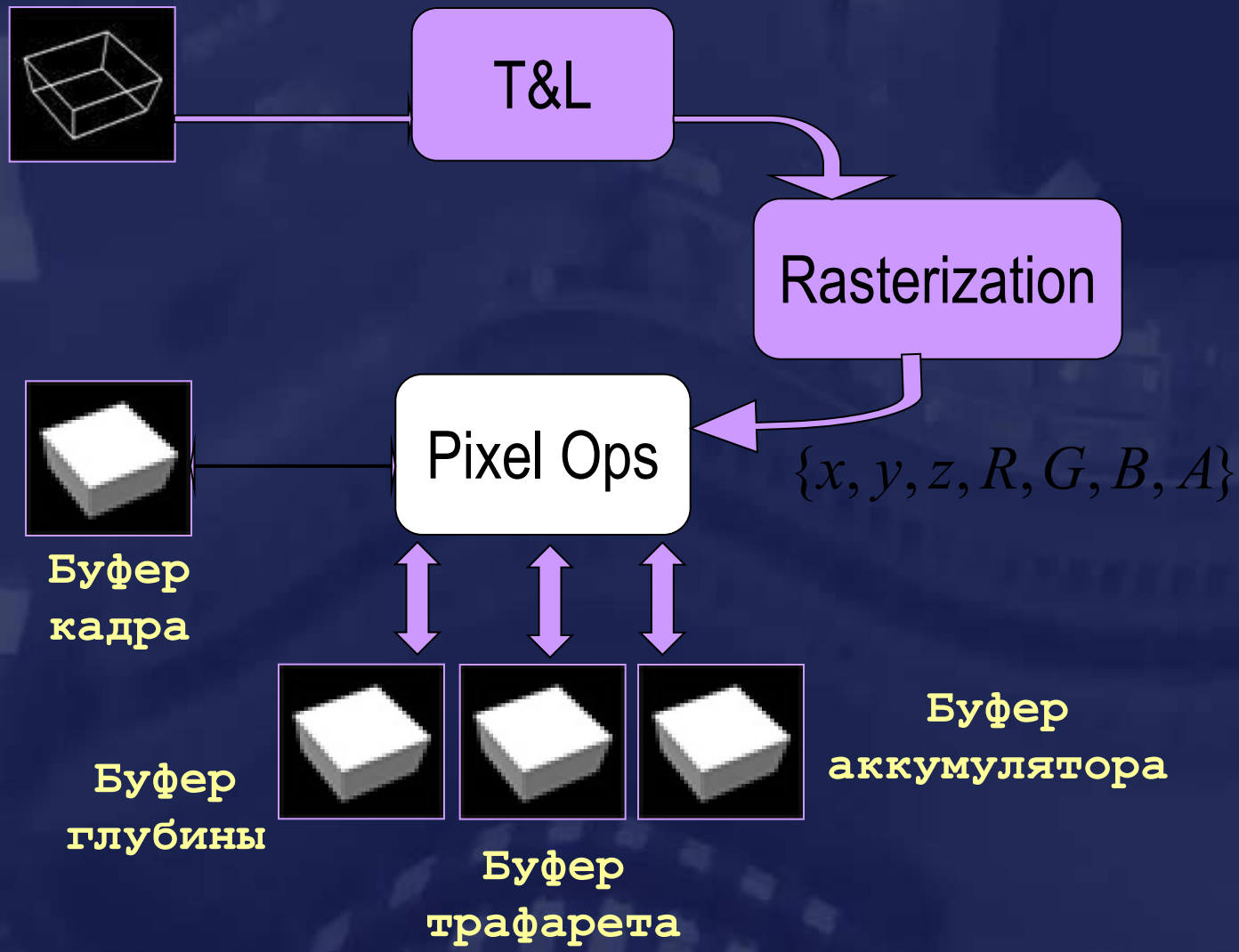


- Работаем с матрицей T точно также как с M и P.

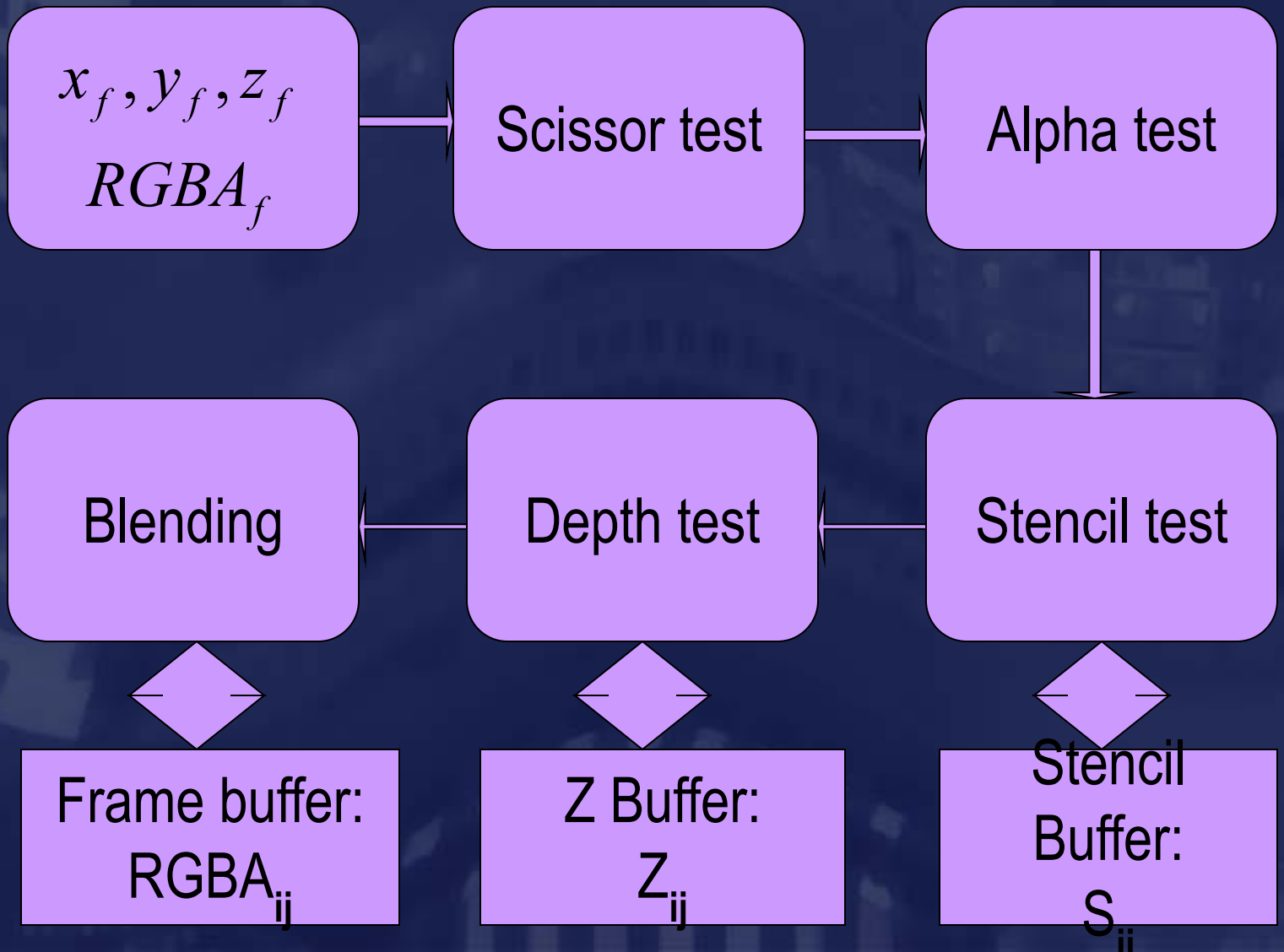
```
glMatrixMode(GL_TEXTURE);
glLoadIdentity();
glMultMatrix(…);
```

- Позволяет существенно изменять вид объекта не изменяя геометрии

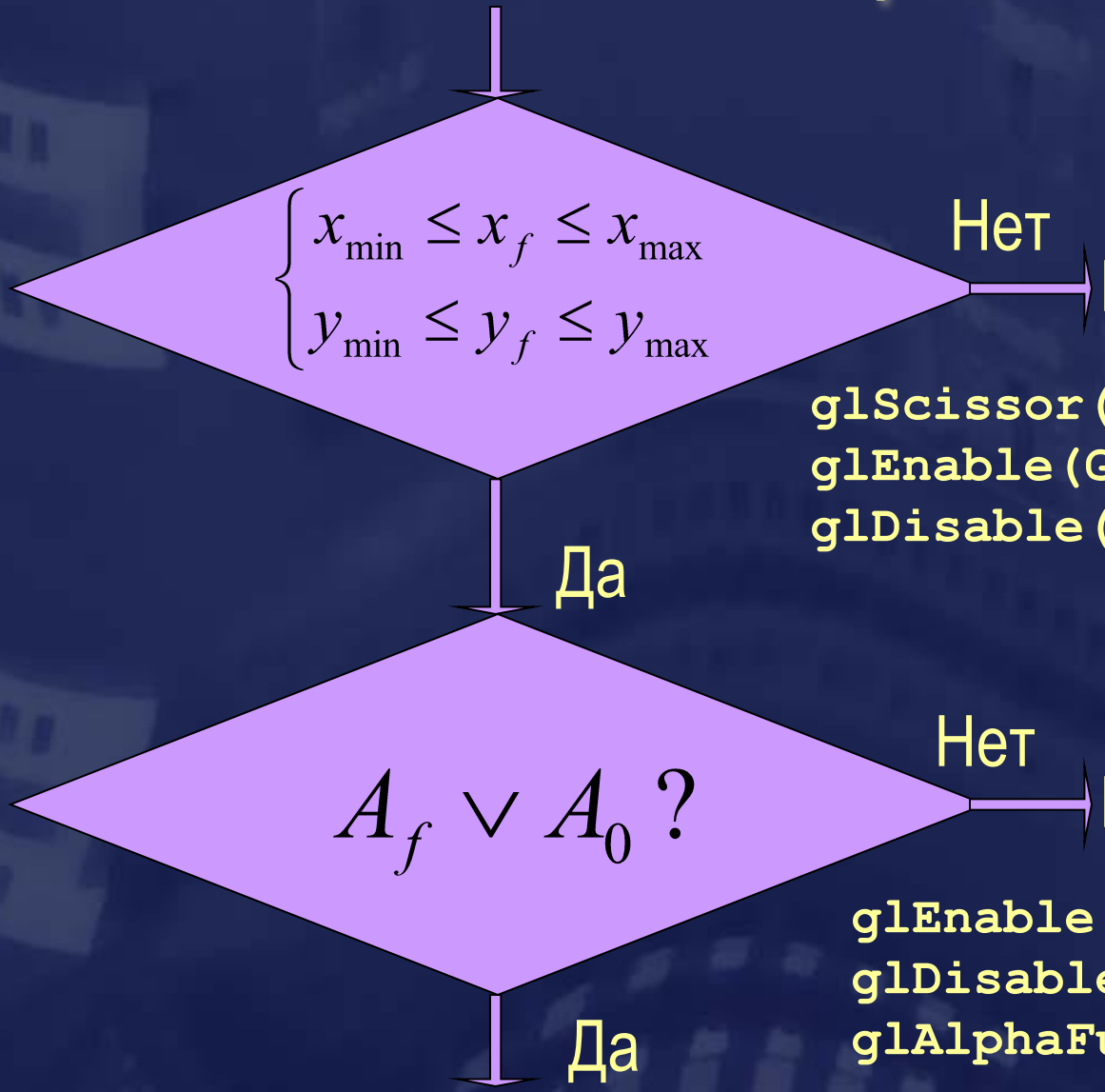
Пиксельные операции



Пиксельные операции. Продолжение.



Scissor & Alpha test




```
glScissor(x,y,w,h);
glEnable(GL_SCISSOR_TEST);
glDisable(GL_SCISSOR_TEST);
```

```
glEnable(GL_SCISSOR_TEST);
glDisable(GL_SCISSOR_TEST);
glAlphaFunc(GL_GREATER,0.5);
```

Смешение цветов

$$C_b = \{R_b, G_b, B_b, A_b\},$$
$$C_f = \{R_f, G_f, B_f, A_f\},$$
$$C_r = D \cdot C_b + S \cdot C_f.$$

- Команды OpenGL:

```
glEnable(GL_BLEND);
glDisable(GL_BLEND);
glBlendFunc(sfactor,dfactor)
```

- glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA):

$$S = \{A_f, A_f, A_f, A_f\}, D = \{1 - A_f, 1 - A_f, 1 - A_f, 1 - A_f\}$$

- glBlendFunc(GL_ONE_MINUS_SRC_ALPHA,GL_SRC_ALPHA):

$$S = \{1 - A_f, 1 - A_f, 1 - A_f, 1 - A_f\}, D = \{A_f, A_f, A_f, A_f\}$$

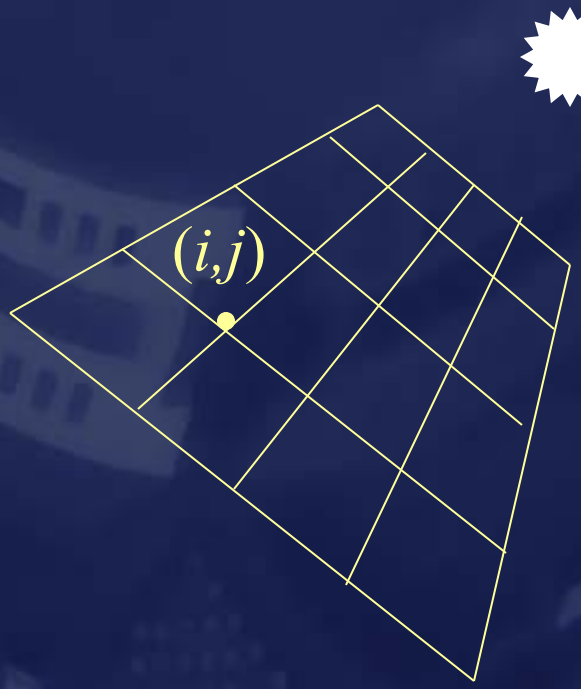
- glBlendFunc(GL_ZERO,GL_SRC_COLOR):

$$S = \{0,0,0,0\}, D = \{R_f, G_f, B_f, A_f\}$$

Карты освещенности (lightmaps)

$$c = e_m + a_m a_s + \sum_{i=0}^{n-1} att_i \cdot spot_i \cdot (a_m a_i + d_m d_i (n \cdot l) + s_m s_i (n \cdot h)^{h_m})$$

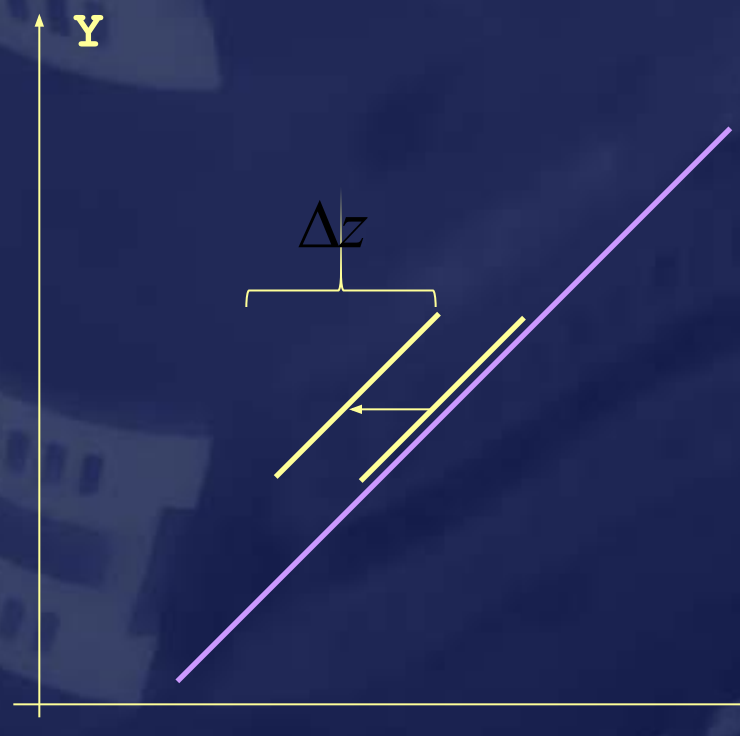
- Карта освещенности: $RGBA_{ij}, 0 \le i < 4, 0 \le j < 4$
- Мультитекстурирование: $R_r = R_1 R_2, G_r = G_1 G_2, B_r = B_1 B_2$



- Возможен предварительный расчет освещения
- Экономия количества примитивов при динамическом освещении
- Экономия текстур при статическом освещении

Смещение вершин

- При использовании буфера глубины могут возникнуть проблемы с рисование граней, лежащих в одной плоскости.

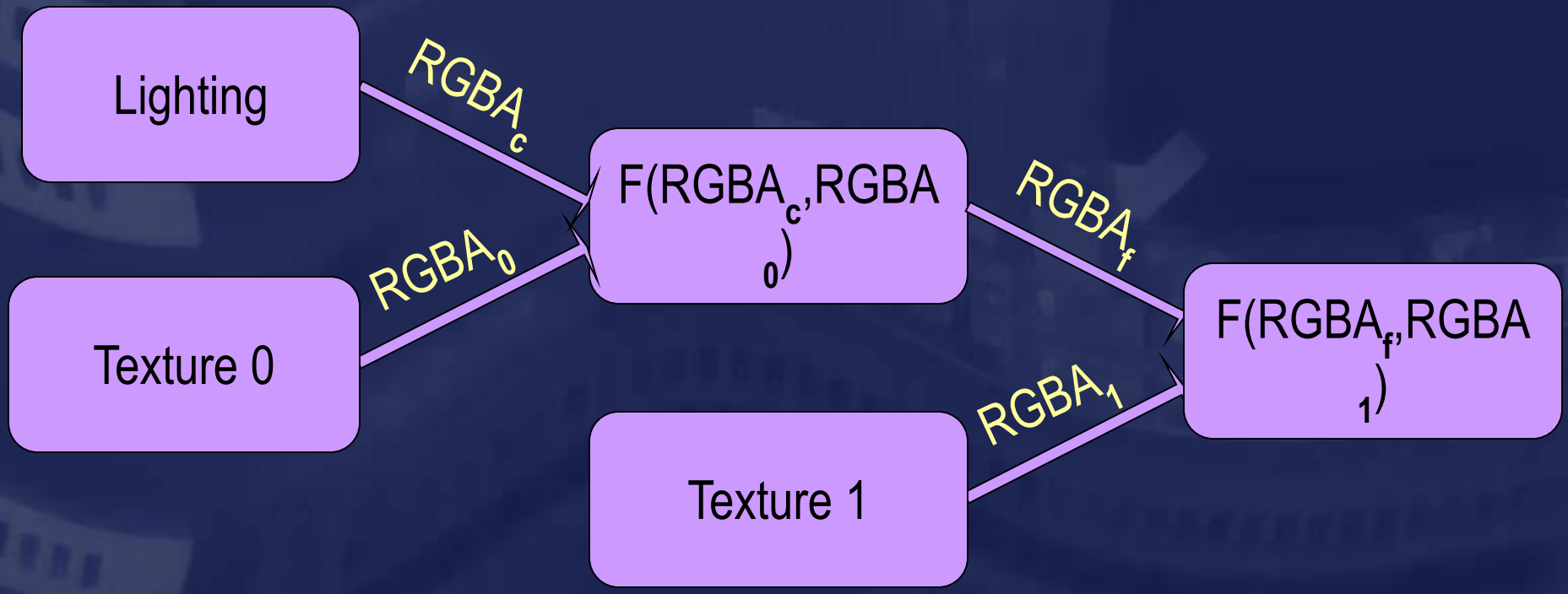


- Возможно задать смещение глубины для каждой вершины

```
void glPolygonOffset(GLfloat factor,
                     GLfloat units);
```

$$z+ = units \cdot r + factor \cdot \Delta z$$

Мультитекстурирование



- Объединение текстуры объекта и текстуры среды

$$RGBA_r = \frac{1}{2}(RGBA_0 + RGBA_1)$$

Расширения OpenGL

- Читаем спецификацию расширения (ARB_multitexture)
- Определяем константы

```
…
#define GL_TEXTURE0_ARB                   0x84C0
#define GL_TEXTURE1_ARB                   0x84C1
#define GL_TEXTURE2_ARB                   0x84C2
#define GL_TEXTURE3_ARB                   0x84C3
…
```

- Определяем указатели на функции

```
…
void (APIENTRY * glMultiTexCoord2d)(GLenum target,
                                    GLdouble s,
                                    GLdouble t);
void (APIENTRY * glActiveTexture)(GLenum target);
…
```

Расширения OpenGL. Продолжение

- Получаем список доступных расширений

```
char *extensions = glGetString(GL_EXTENSIONS);
```

Расширения OpenGL. Часть 3.

- Получаем указатели на функции

```
…
 glActiveTexture = wglGetProcAddress("glActiveTextureARB");
 glMultiTexCoord2d =
wglGetProcAddress("glMultiTexCoord2dARB");
…
```

- Задаем текстурные объекты для каждого текстурного блока

```
…
(*glActiveTexture)(GL_TEXTURE0_ARB);
glEnable(GL_TEXTURE_2D);
glBindTexture(…);
…
(*glActiveTexture)(GL_TEXTURE1_ARB);
glEnable(GL_TEXTURE_2D);
glBindTexture(…);
…
```

Расширения OpenGL. Часть 4.

- Задаем смешение цвета освещения и цвета первой текстуры

```
…
(*glActiveTexture)(GL_TEXTURE0_ARB);
glTexEnvi(…);
…
```

- Задаем смешение цвета первого блока и второй текстуры

```
…
(*glActiveTexture)(GL_TEXTURE1_ARB);
glTexEnvi(…);
…
```

- Задаем текстурные координаты для каждого текстурного блока

```
…
(*glMultiTexCoord2d)(GL_TEXTURE0_ARB,0.5,0.5);
(*glMultiTexCoord2d)(GL_TEXTURE1_ARB,0.2,0.3);
…
```

Полупрозрачные объекты

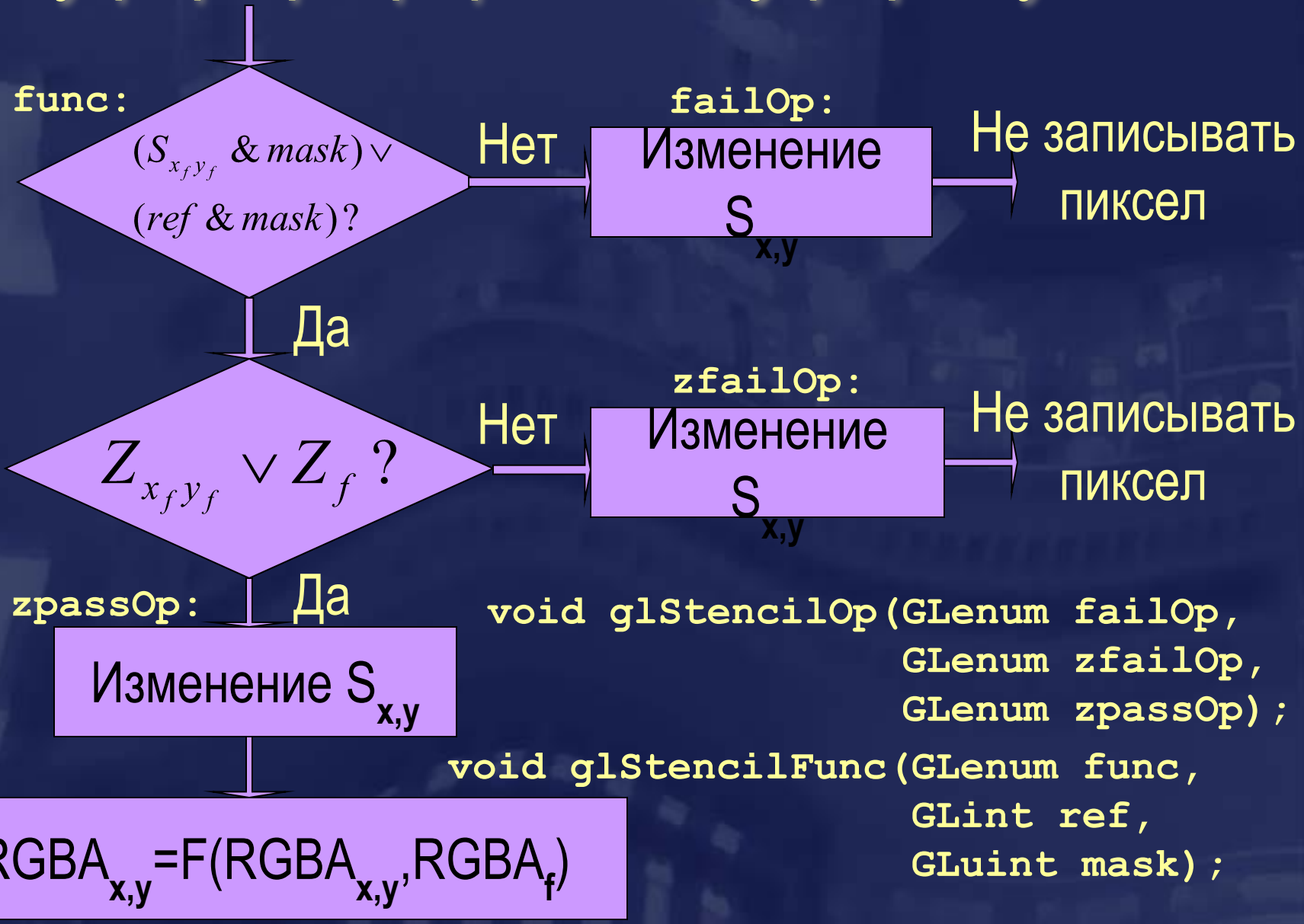
- Полупрозрачная грань - грань через часть пикселов которой видно лежащие за ней грани.

- Полупрозрачные грани необходимо выводить в порядке back-to-front. Применение метода Z-буфера ведет к визуальным артефактам -“глюкам” :)

- Для выпуклых объектов можно выводить сначала нелицевые грани, а затем - лицевые.

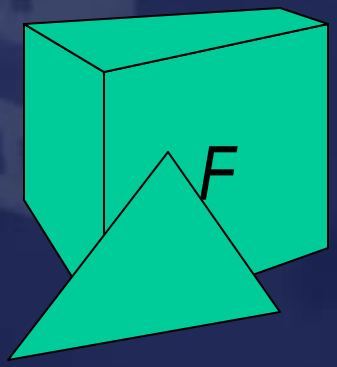
- Объекты можно выводить методом художника - сортировать по убыванию координат Z центров.

Буфер трафарета и буфер глубины



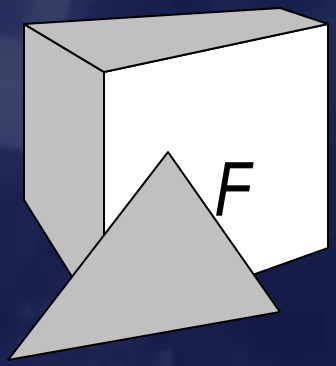

```
void glStencilOp(GLenum failOp,
                 GLenum zfailOp,
                 GLenum zpassOp);
void glStencilFunc(GLenum func,
                   GLint ref,
                   GLuint mask);
```

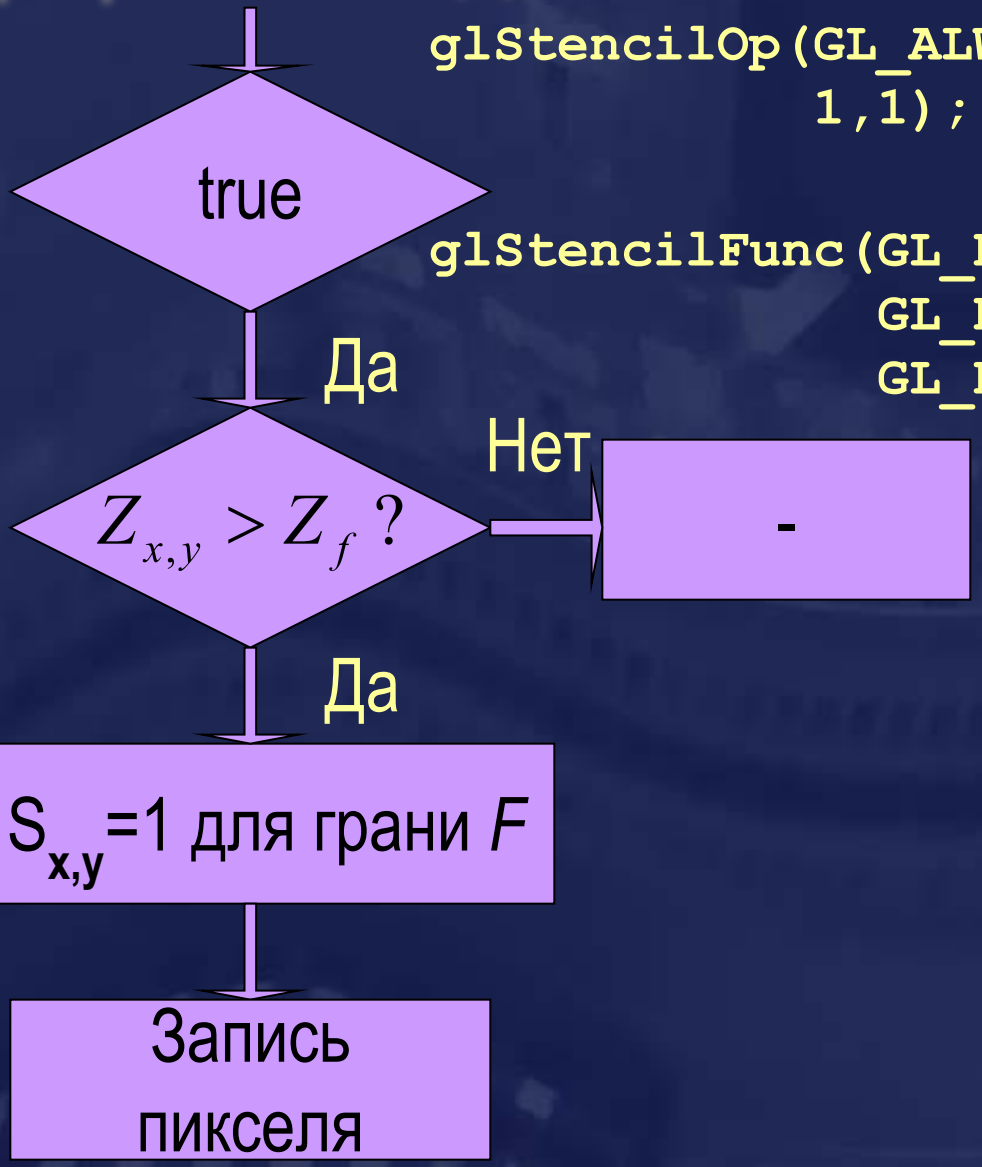
Буфер трафарета: видимость пикселей



$S_{x,y}$ == 1

$S_{x,y}$ == 0





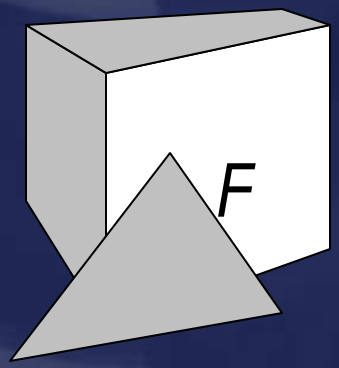

```
glStencilOp(GL_ALWAYS,
            1,1);
```

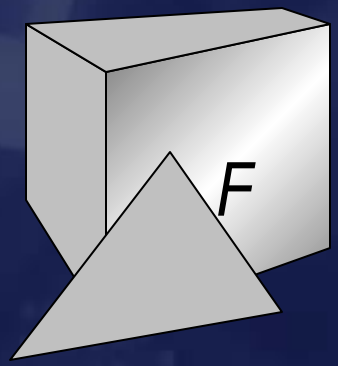
```
glStencilFunc(GL_KEEP,
              GL_KEEP,
              GL_REPLACE);
```

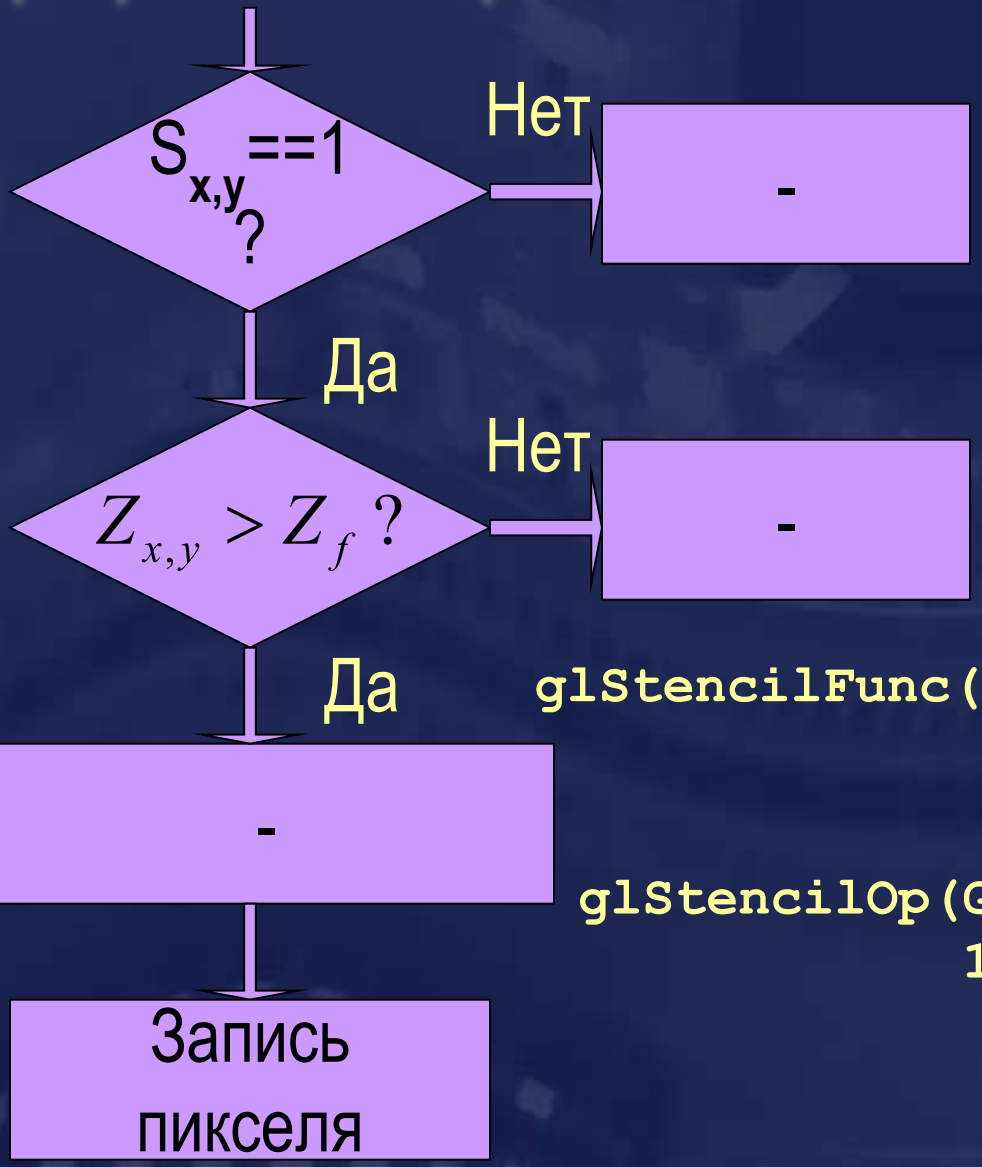
Буфер трафарета: обработка пикселей

$S_{x,y}$ == 1

$S_{x,y}$ == 0



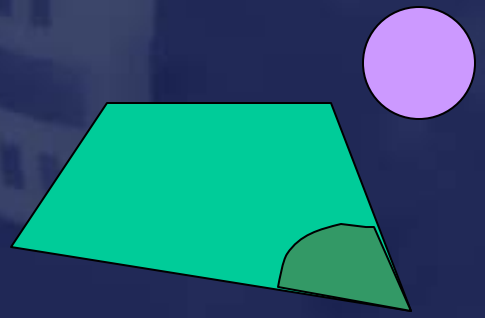




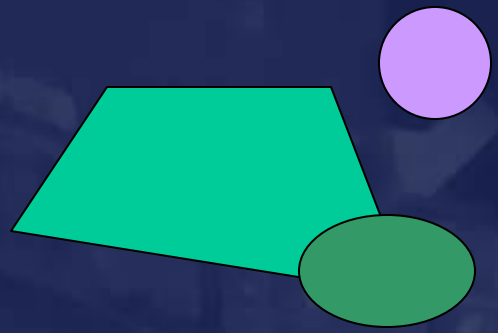

```
glStencilFunc(GL_KEEP,
              GL_KEEP,
              GL_KEEP);
glStencilOp(GL_EQUAL,
            1,1);
```

Буфер трафарета: тени и отражения

- Отбрасывание тени на плоскую грань

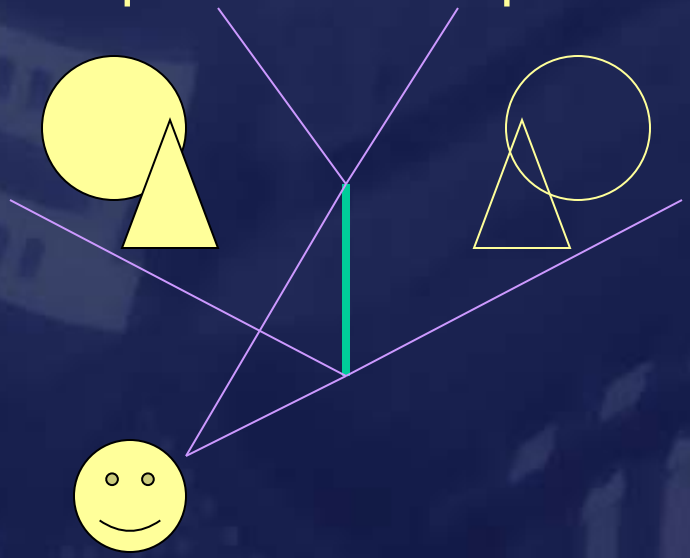


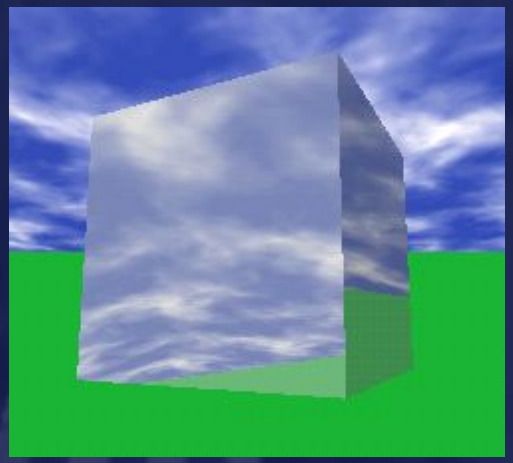
Правильно

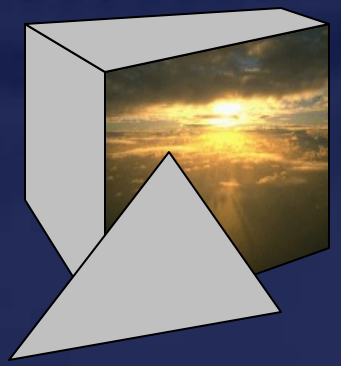


Неправильно

- Отражения и порталы

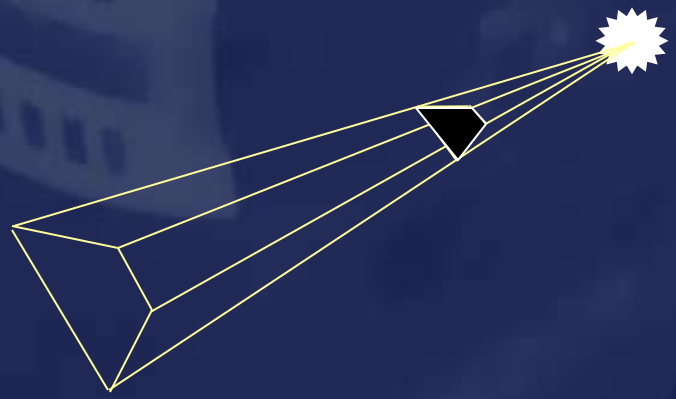






Буфер трафарета: теневые объемы

- Строим теневой объем

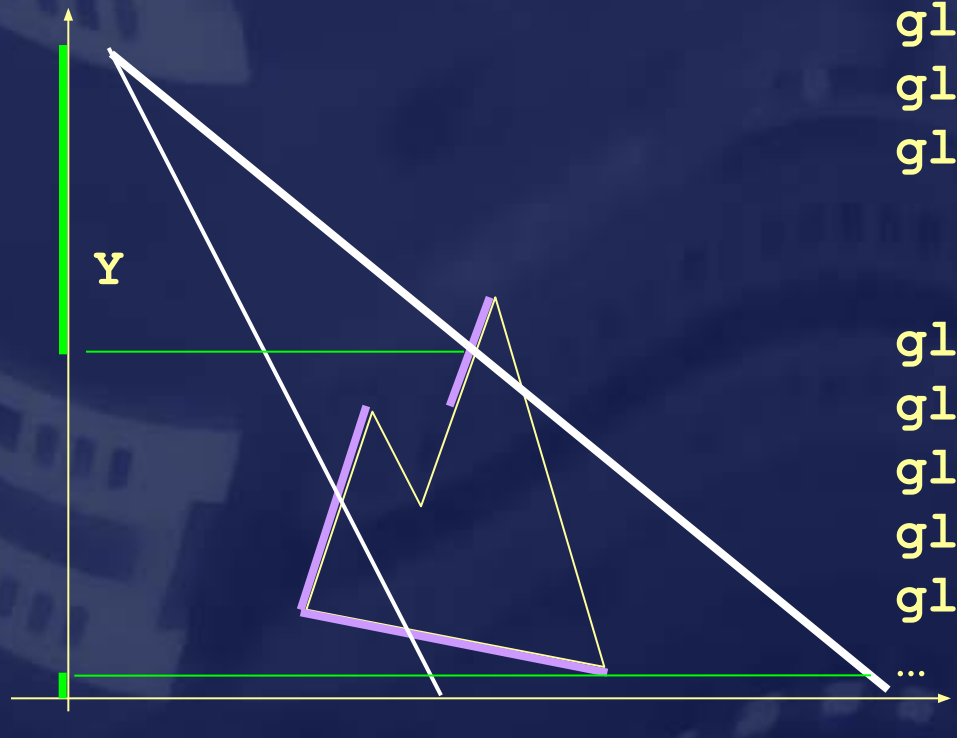


- Рисуем затеняемый объект со включенным Z-буфером


```
…
glEnable(GL_DEPTH_TEST);
glCallList(object);
…
```

Теневые объемы. Продолжение.

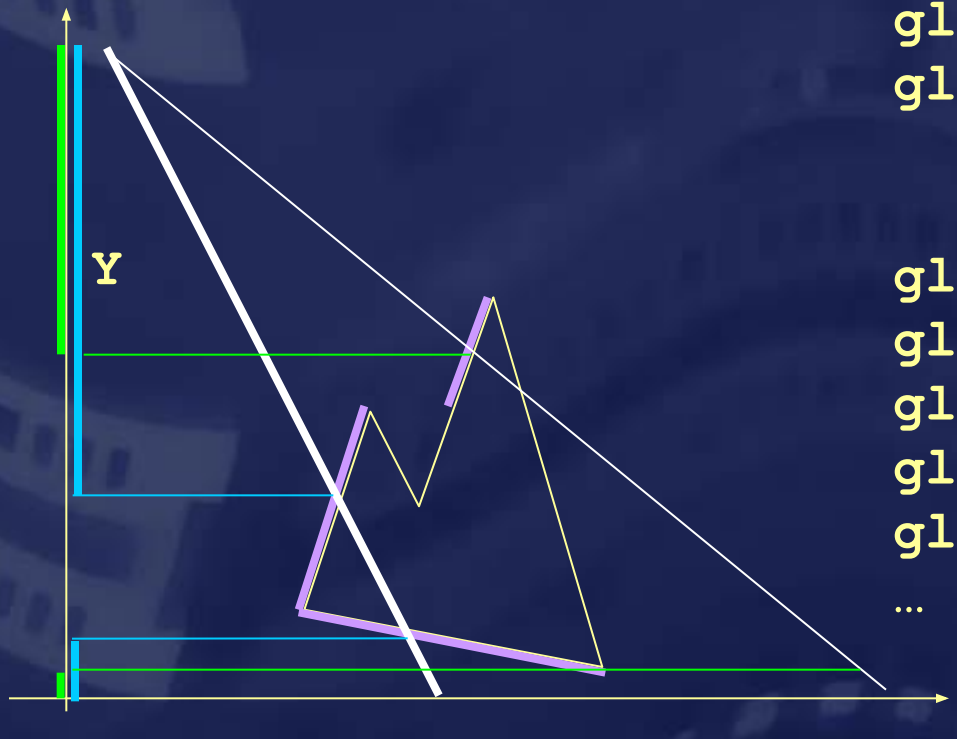
- Рисуем **нелицевые** грани теневого объема




```
…
glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_ALWAYS,1,1);
glStencilOp(GL_KEEP,
            GL_KEEP,
            GL_REPLACE);
glColorMask(0,0,0,0);
glDepthMask(0);
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
glCallList(shadow);
…
```

Теневые объемы. Часть 3.

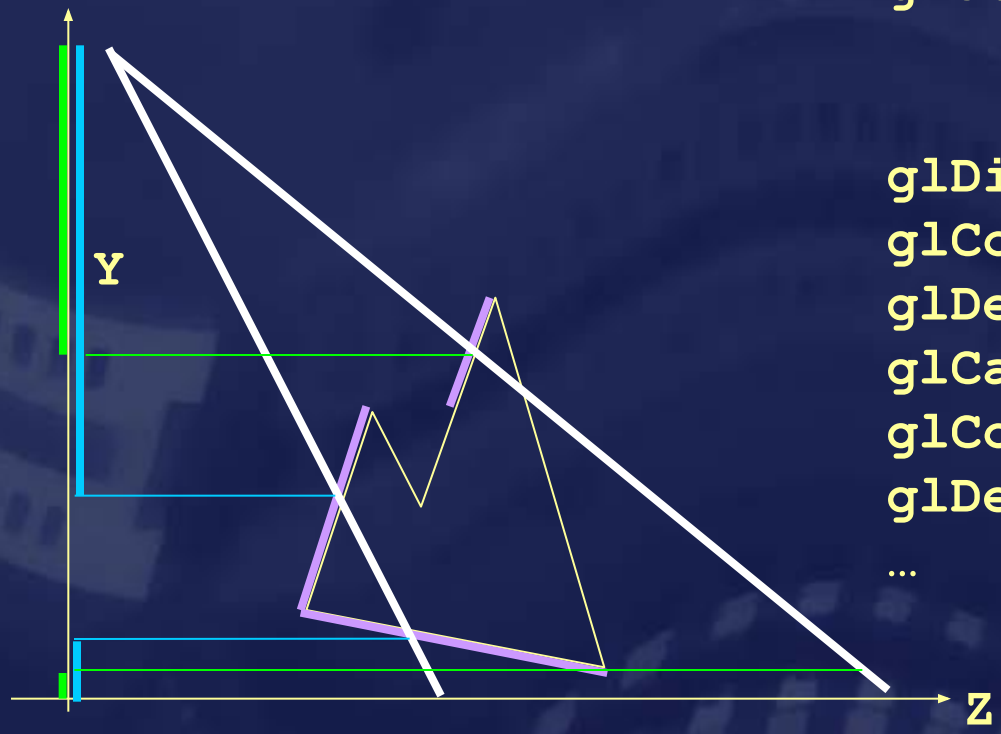
- Рисуем **лицевые** грани теневого объема




```
…
glStencilFunc(GL_ALWAYS,1,1);
glStencilOp(GL_KEEP,
            GL_KEEP,
            GL_INVERT);
glEnable(GL_CULL_FACE);
glCullFace(GL_FRONT);
glCallList(shadow);
glColorMask(1,1,1,1);
glDepthMask(1);
…
```

Теневые объемы. Часть 4.

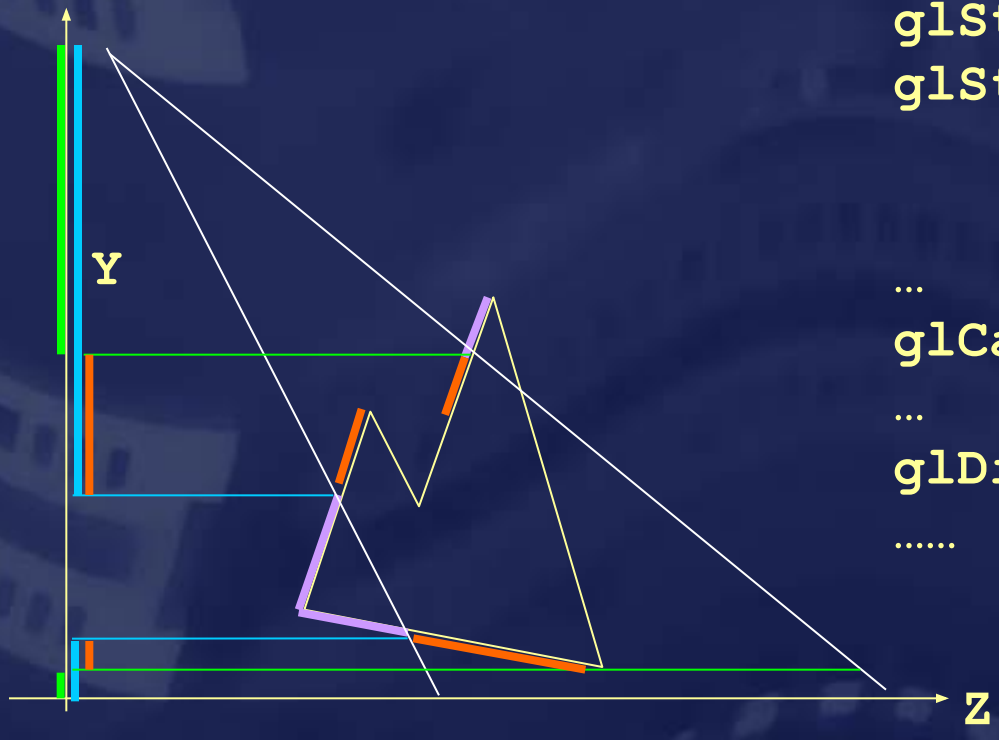
- Возможен вывод граней теневого объема в произвольном порядке.




```
…
glStencilFunc(GL_ALWAYS,1,1);
glStencilOp(GL_KEEP,
            GL_KEEP,
            GL_INVERT);
glDisable(GL_CULL_FACE);
glColorMask(0,0,0,0);
glDepthMask(0);
glCallList(shadow);
glColorMask(1,1,1,1);
glDepthMask(1);
…
```

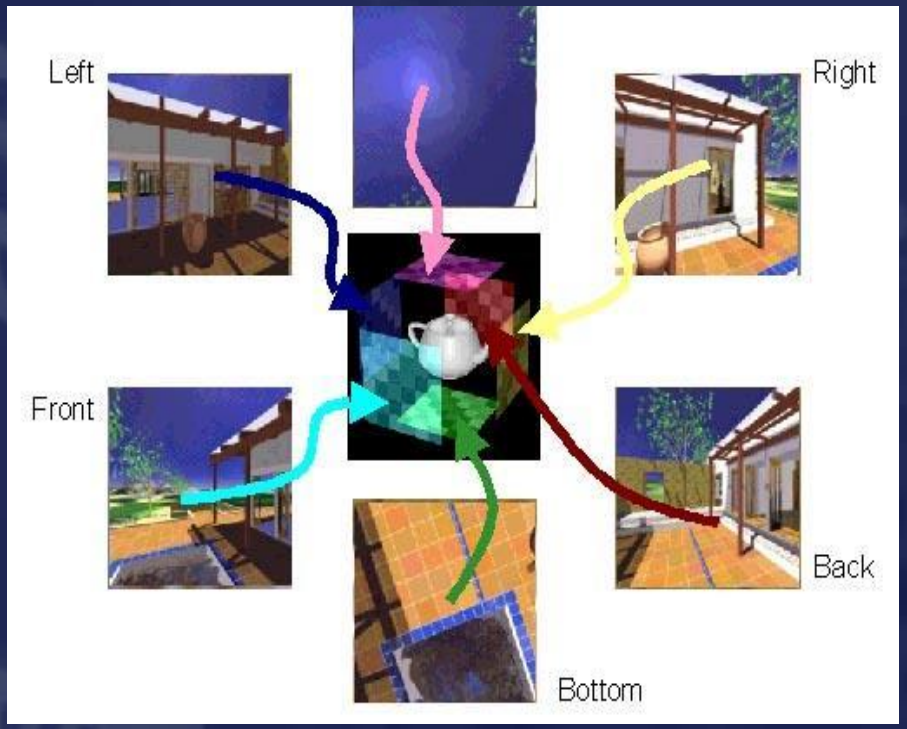
Теневые объемы. Часть 5.

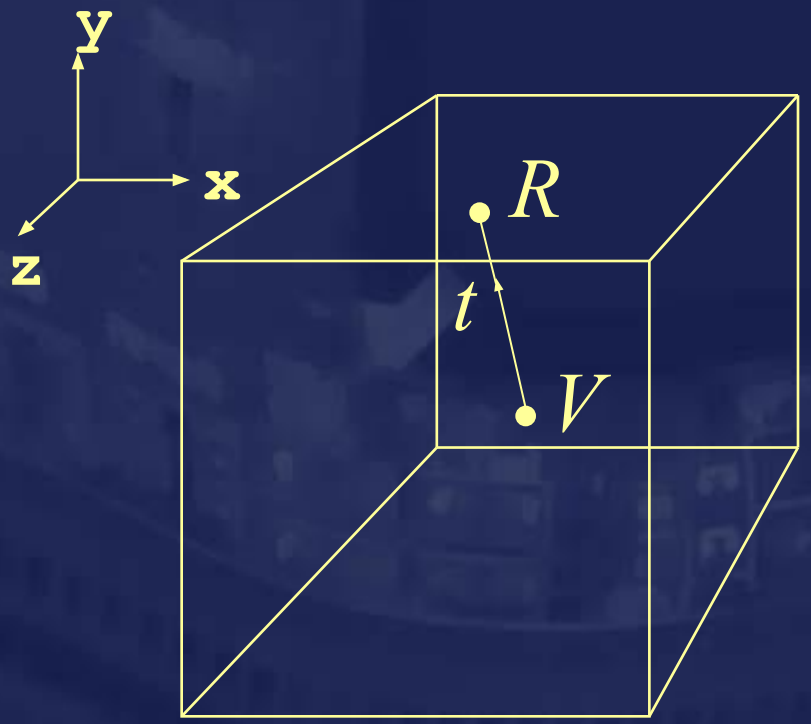
- Рисуем затененную часть объекта




```
…
glStencilFunc(GL_EQUAL,1,1);
glStencilOp(GL_KEEP,
            GL_KEEP,
            GL_KEEP);
…
glCallList(object);
…
glDisable(GL_STENCIL_TEST);
……
```

Кубические карты: определение.

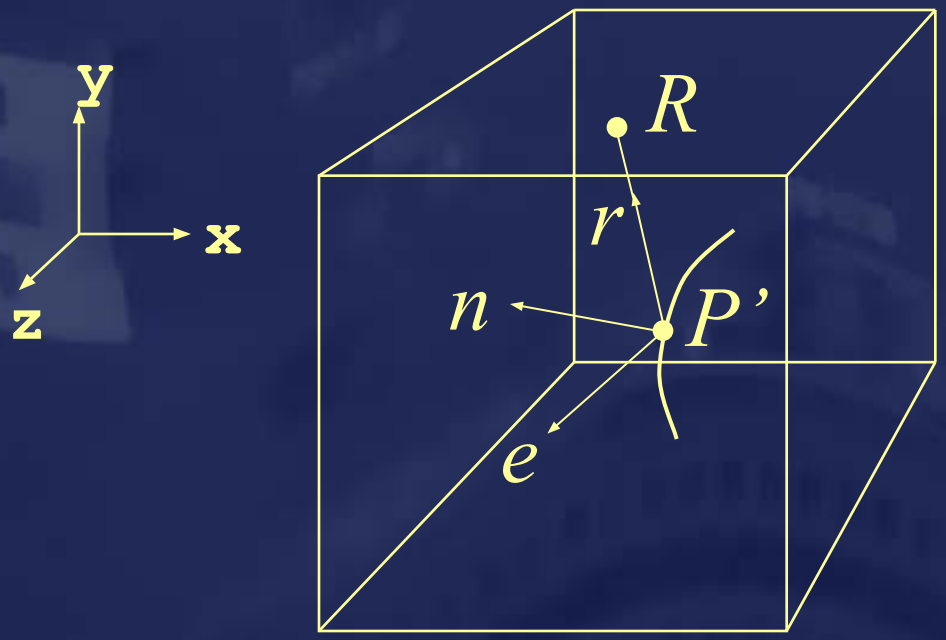




- С вершиной V связано три текстурные координаты (p,q,r).
- Точка R = R(t), t = (p,q,r) определяет одну из шести текстур и текстурные координаты. (p,q,r)->RGBA

Расширение "GL_EXT_cube_map"

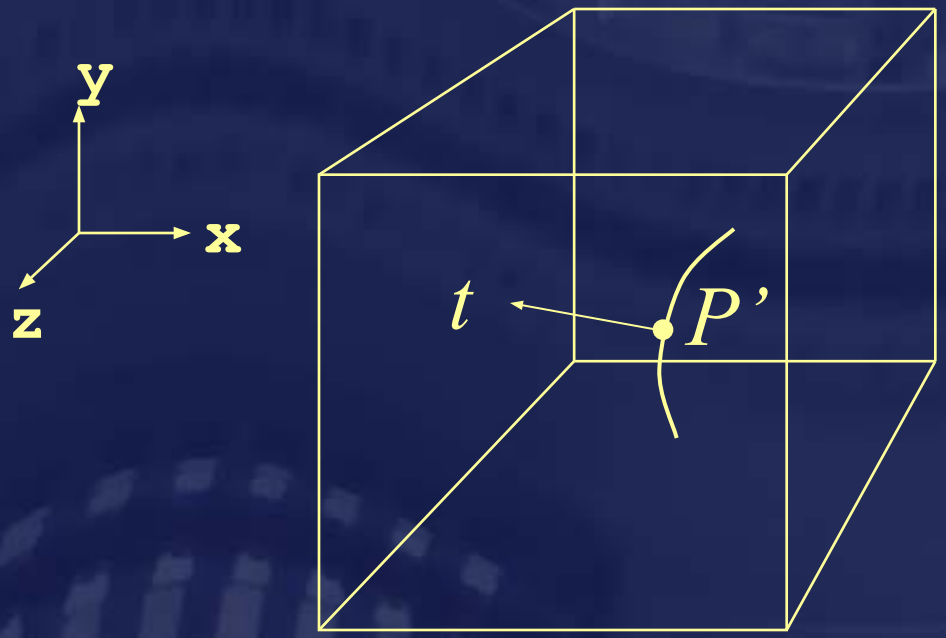
Кубические карты среды.



- Необходимо применить к текстурным координатам преобразование, обратное модельно-видовому!

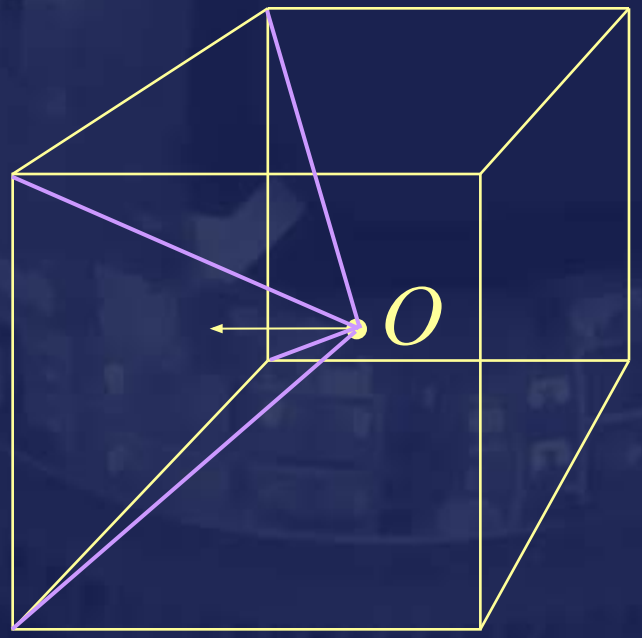
Кубические карты освещения.

- Пусть источники освещения находятся далеко от объекта. Тогда освещение в точке объекта зависит только от нормали.
- Заранее запишем результат расчета освещенности для нормали *n* в элемент кубической текстуры с координатами *n*.



Построение отражающего объекта.

- Переносим наблюдателя в центр отражающего объекта и строим изображение, полученное при взгляде в направлении нормали к одной из сторон.



- Сохраняем изображение на экране как текстуру!

```
void glCopyTexImage2D(
     GLenum target, GLint level,
     GLenum internalFormat,
     GLint x, GLint y, Glsizei width, GLsizei height,
     GLint border );
```

- Рисуем объект с кубическими картами отражения.

cg@cs.msu.su