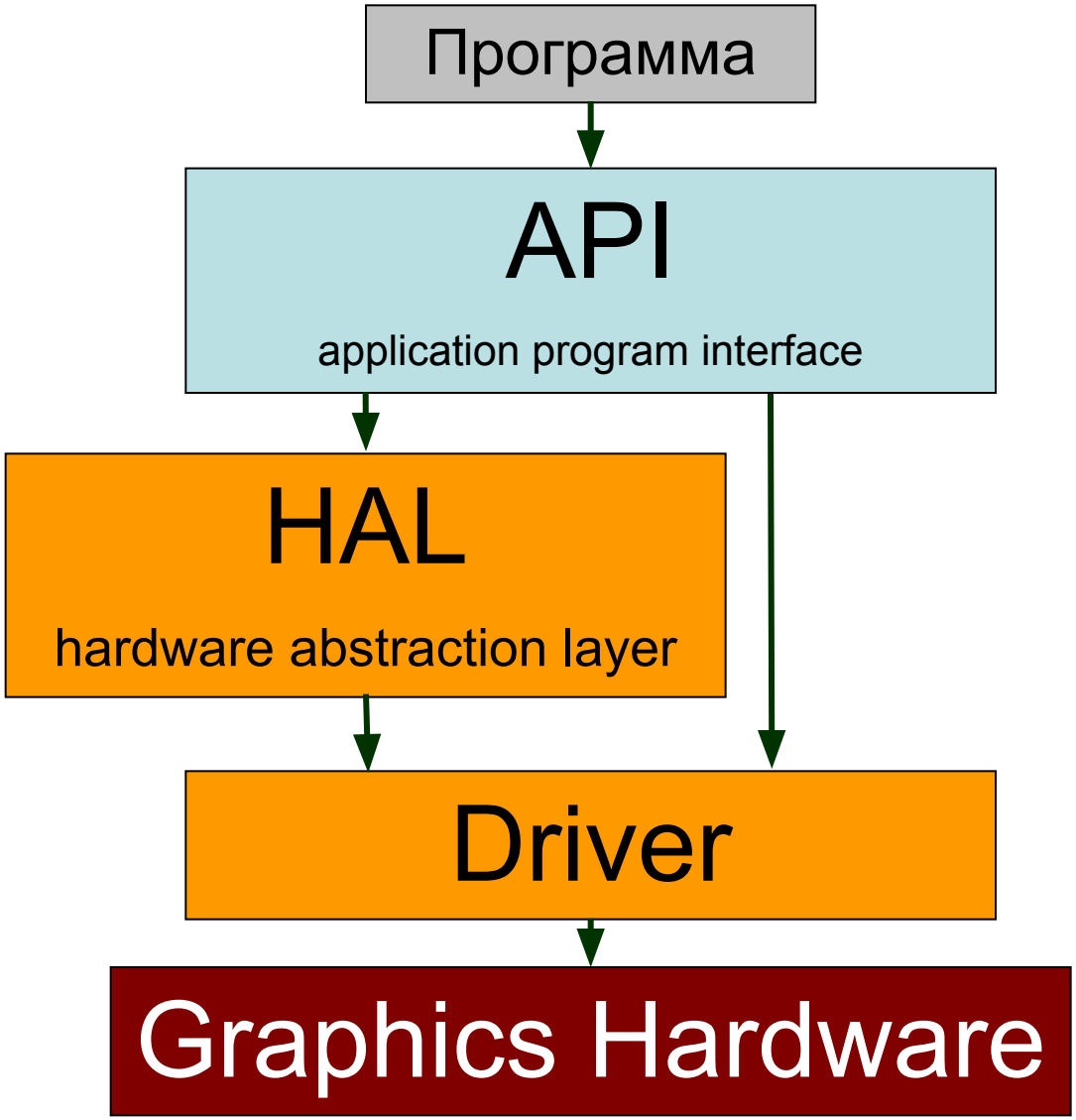
Компьютерная графика реального времени.

URL: http://www.school30.spb.ru/cgsg/cgc/
E-mail: CGSG@yandex.ru

Структура взаимодействия приложения с аппаратурой



- **Open GL** – open graphics library (SGI, 90-е годы, версии 1.0-4.2)
  - GLUT - OpenGL Utility Toolkit (http://www.opengl.org/resources/libraries/glut/)
  - GLEW - OpenGL Extension Wrangler Library (http://glew.sourceforge.net/)
  - GLM - OpenGL Mathematics (http://glm.g-truc.net/)
  - . . .

- **Microsoft Direct3D** – часть MS DirectX (1992 RenderMorphics, версии 2.0-11.0)
  - D3DX – retained mode toolkit
  - XNA - Xbox New Architecture (http://msdn.microsoft.com/ru-ru/xna/)

Инициализация OpenGL


```
PIXELFORMATDESCRIPTOR pfd = {0};

pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);
pfd.dwFlags =
  PFD_SUPPORT_OPENGL | PFD_SUPPORT_GDI |PFD_DOUBLEBUFFER;
pfd.iPixelType = PFD_TYPE_RGBA;
pfd.cColorBits = 32;
pfd.cDepthBits = 24;
pfd.cStencilBits = 8;

i = ChoosePixelFormat(hDC, &pfd);
DescribePixelFormat(hDC, i, sizeof(PIXELFORMATDESCRIPTOR), &pfd);

SetPixelFormat(hDC, i, &pfd);

hRC = wglCreateContext(hDC);
wglMakeCurrent(hDC, hRC);
```

Инициализация OpenGL: GLUT


```
#include <glut.h>

  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(0, 0);
  glutCreateWindow("OpenGL Sample");

  glutDisplayFunc(Display);
  glutIdleFunc(Idle);
  glutReshapeFunc(Reshape);
  glutKeyboardFunc(Keyboard);

  MyInit();

  glutMainLoop();
```

GLUT: Callbacks

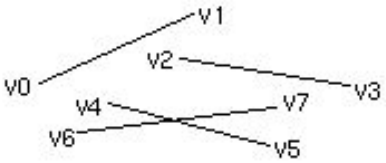

```
void Reshape( int w, int h )
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60, (double)w / h, 1, 100);
  gluLookAt(5, 5, 5,   0, 0, 0,   0, 1, 0);
  glMatrixMode(GL_MODELVIEW);
}
void Display( void )
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  . . .
  glFinish();
  glutSwapBuffers();
}
void Idle( void )
{
  glutPostRedisplay();
}
void Keyboard( char key, int x, int y )
{
  if (key == 27)
    exit(0);
}
```

Инициализация Direct3D

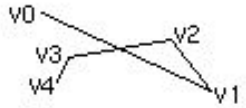

```
IDirect3D9 *D3D;
IDirect3DDevice9 *D3DDev;
D3DPRESENT_PARAMETERS PresParams = {0};
D3DDISPLAYMODE dm;

if ((D3D = Direct3DCreate9(D3D_SDK_VERSION)) == NULL)
  return FALSE;

D3D->GetAdapterDisplayMode(D3DADAPTER_DEFAULT, &dm);

PresParams.Windowed = TRUE;
PresParams.hDeviceWindow = hWnd;
PresParams.BackBufferFormat = dm.Format;
PresParams.BackBufferCount = 1;
PresParams.PresentationInterval = D3DPRESENT_INTERVAL_IMMEDIATE;
PresParams.SwapEffect = D3DSWAPEFFECT_DISCARD;
PresParams.BackBufferWidth = 640;
PresParams.BackBufferHeight = 480;
PresParams.EnableAutoDepthStencil = TRUE;
PresParams.AutoDepthStencilFormat = D3DFMT_D24S8;

if (D3D->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
      hWnd, D3DCREATE_HARDWARE_VERTEXPROCESSING,
      &PresParams, &D3DDev) != S_OK)
{
  D3D->Release();
  return FALSE;
}
```

Примитивы OpenGL

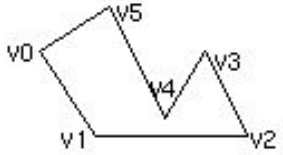
GL_POINTS



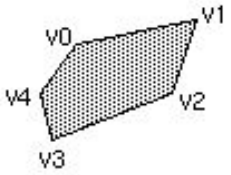
GL_LINES



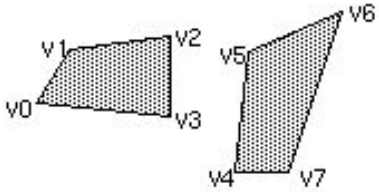
GL_LINE_STRIP



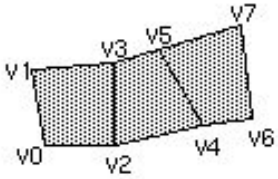
GL_LINE_LOOP



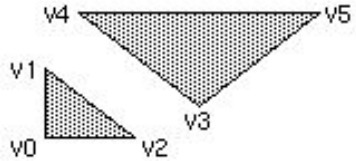
GL_POLYGON



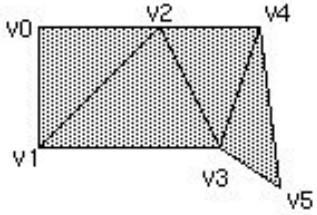
GL_QUADS



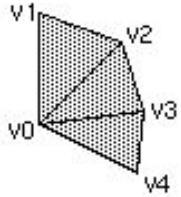
GL_QUAD_STRIP



GL_TRIANGLES



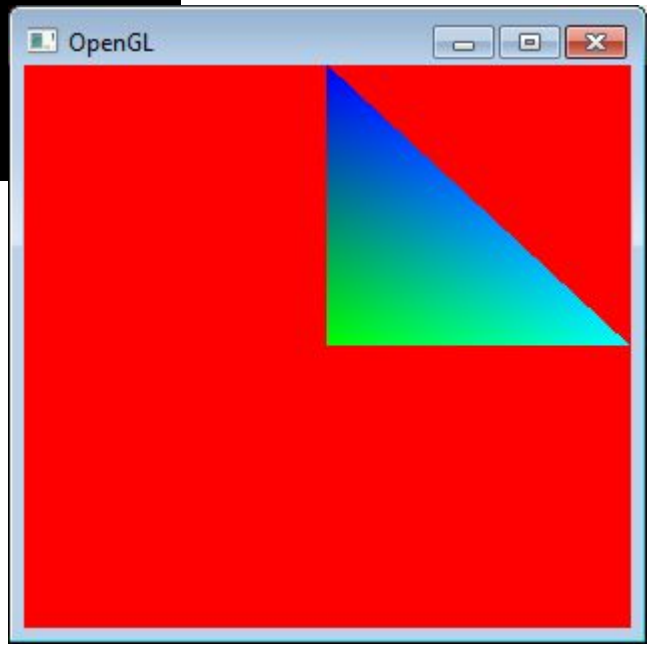
GL_TRIANGLE_STRIP



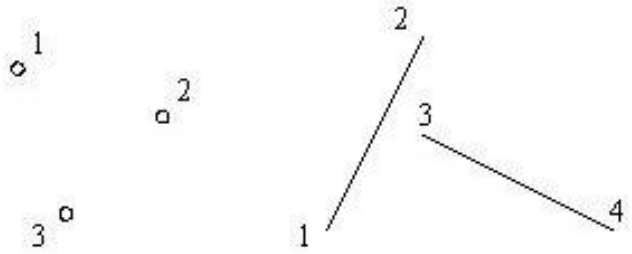
GL_TRIANGLE_FAN

Вывод OpenGL


```
glClearColor(1, 0, 0, 1);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glBegin(GL_TRIANGLES);
  glColor3d(0, 1, 0);
  glVertex3d(0, 0, 0);
  glColor3d(0, 1, 1);
  glVertex3d(1, 0, 0);
  glColor3d(0, 0, 1);
  glVertex3d(0, 1, 0);
glEnd();

glFinish();
glutSwapBuffers();
```

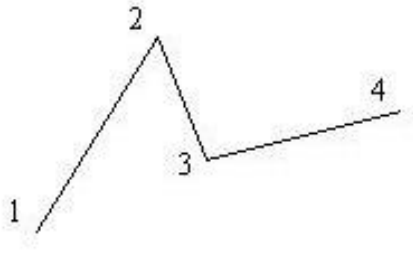


Примитивы Direct3D

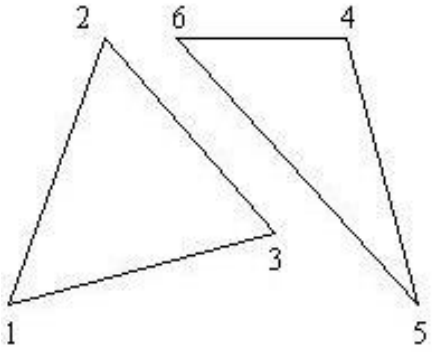


D3DPT_POINTLIST

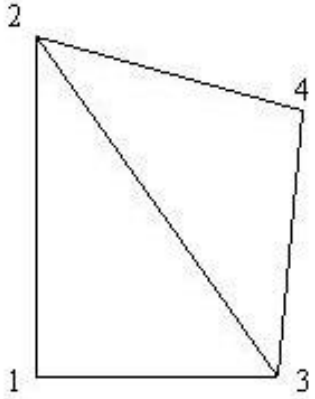
D3DPT_LINELIST



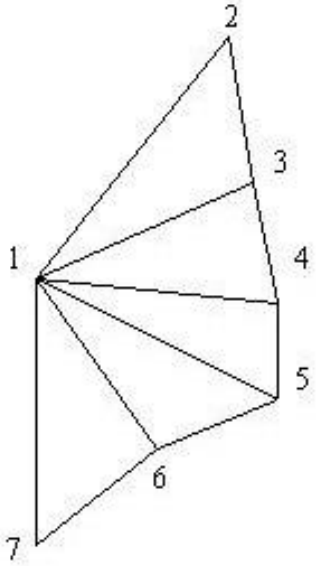
D3DPT_LINESTRIP



D3DPT_TRIANGLELIST



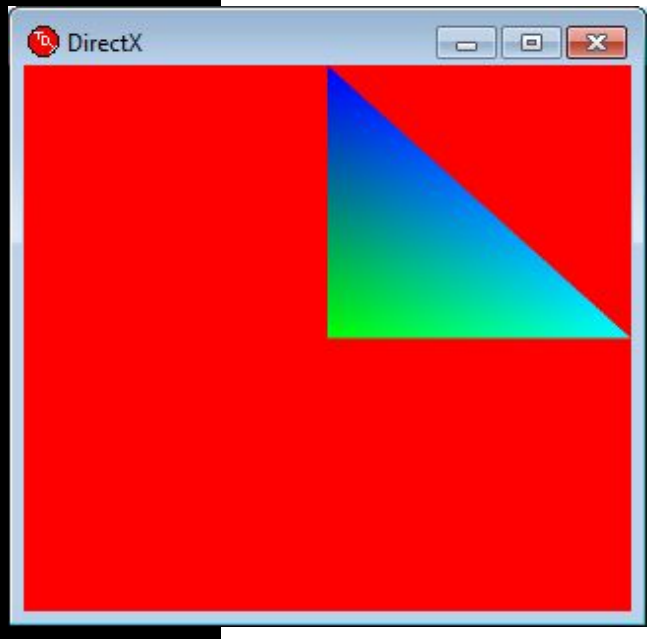
D3DPT_TRIANGLESTRIP



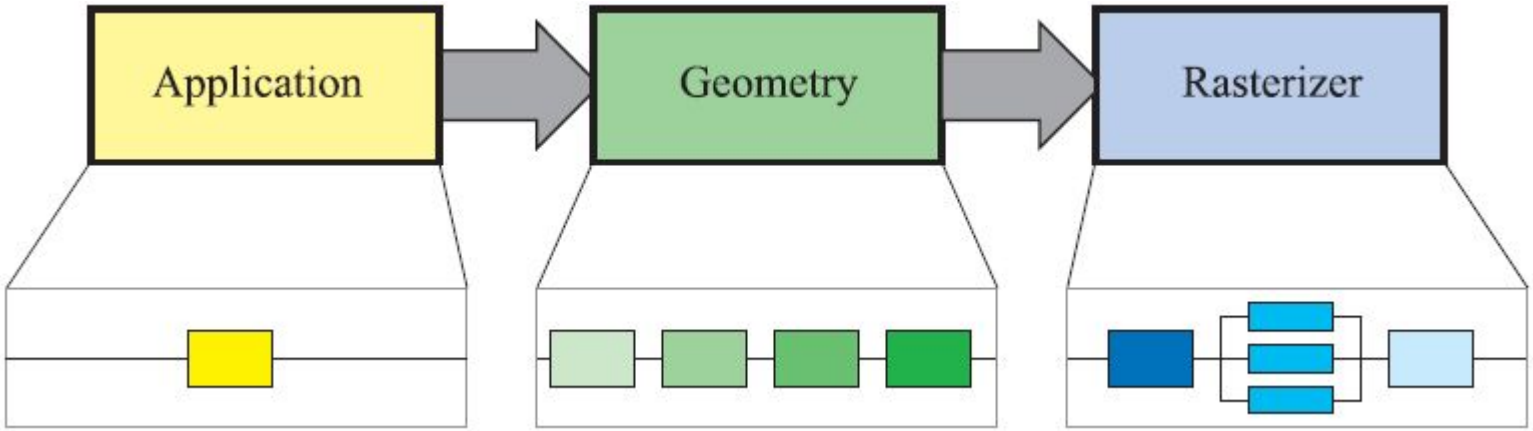
D3DPT_TRIANGLEFAN

Вывод Direct3D


```
D3DDev->SetFVF(D3DFVF_XYZ | D3DFVF_DIFFUSE);
D3DDev->Clear(0, NULL,
  D3DCLEAR_TARGET | D3DCLEAR_STENCIL | D3DCLEAR_ZBUFFER,
  D3DCOLOR_XRGB(255, 0, 0), 1.0, 0);

D3DDev->BeginScene();

D3DDev->SetRenderState(D3DRS_LIGHTING, FALSE);
D3DDev->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);

struct VERTEX
{
  FLOAT X, Y, Z;
  DWORD Color;
} pnts[] =
{
  {0, 0, 0, D3DCOLOR_XRGB(0, 255, 0)},
  {1, 0, 0, D3DCOLOR_XRGB(0, 255, 255)},
  {0, 1, 0, D3DCOLOR_XRGB(0, 0, 255)},
};

D3DDev->DrawPrimitiveUP(D3DPT_TRIANGLELIST,
  1, pp, sizeof(VERTEX));

D3DDev->EndScene();
D3DDev->Present(NULL, NULL, NULL, NULL);
```

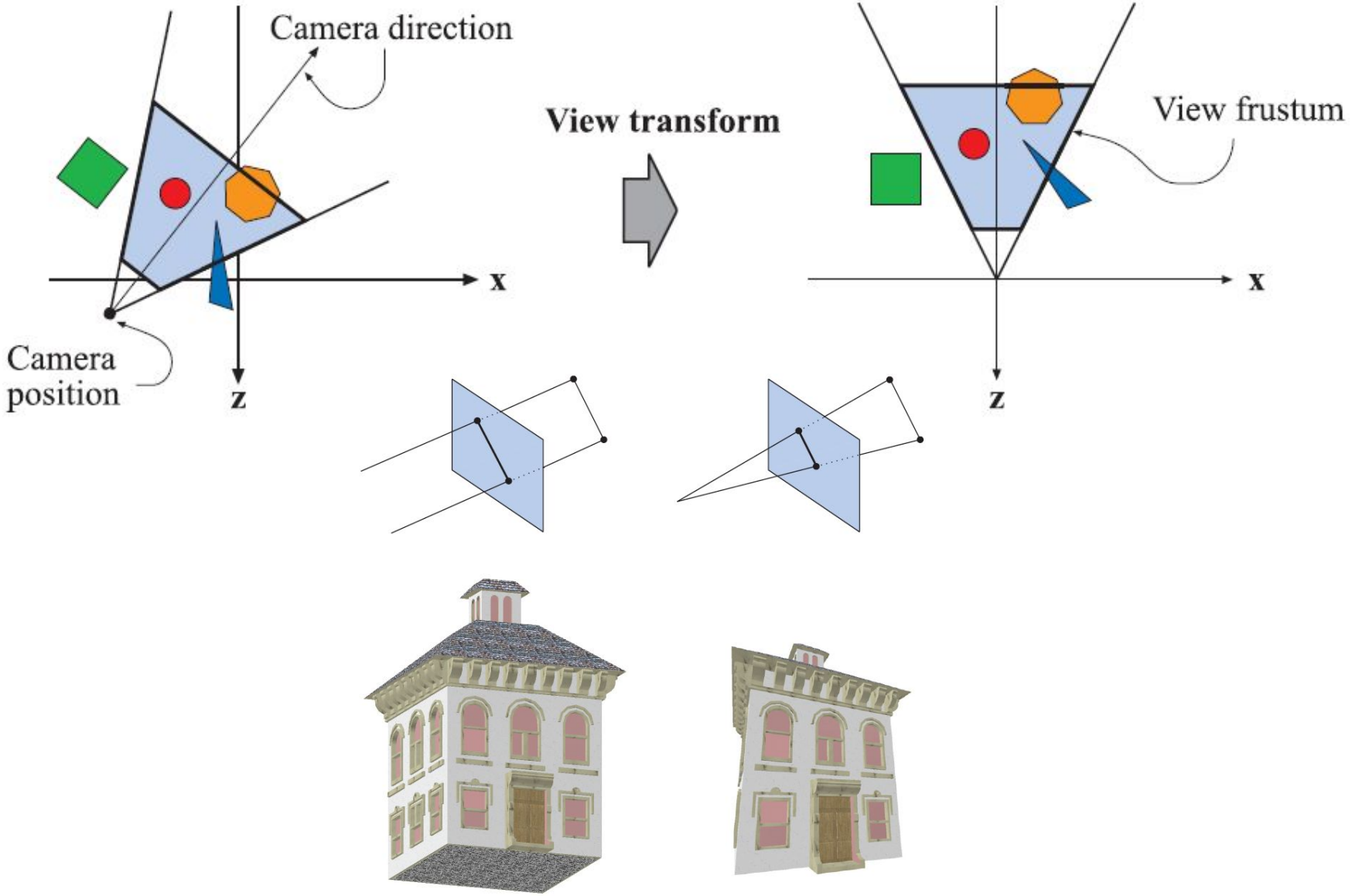


Общая архитектура

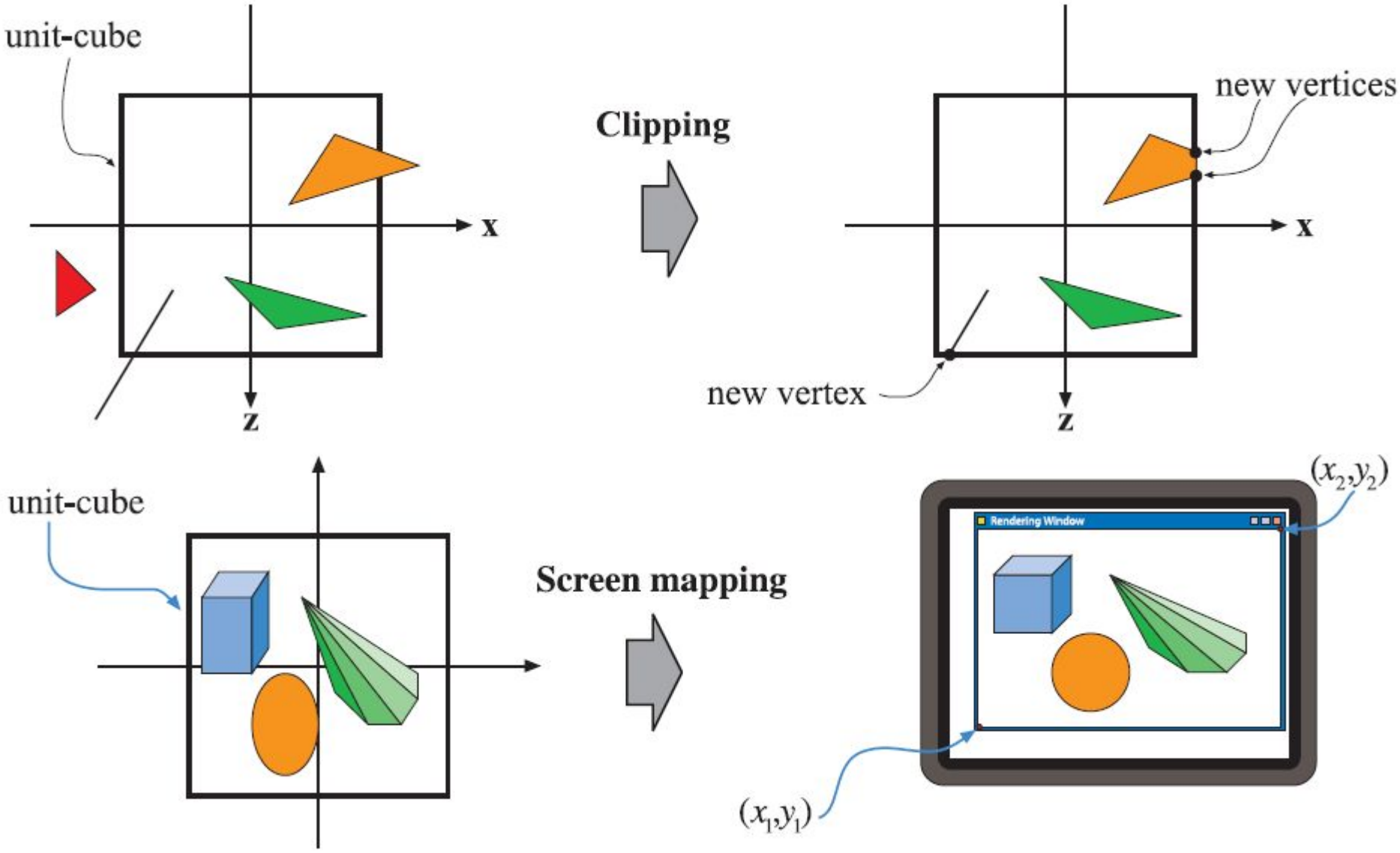


Геометрическая фаза

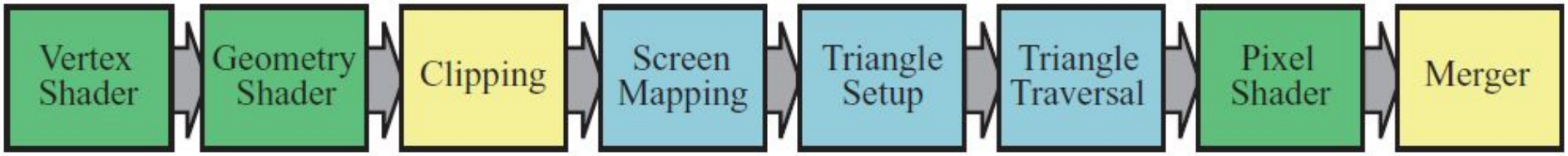
Видовые преобразования & проецирование



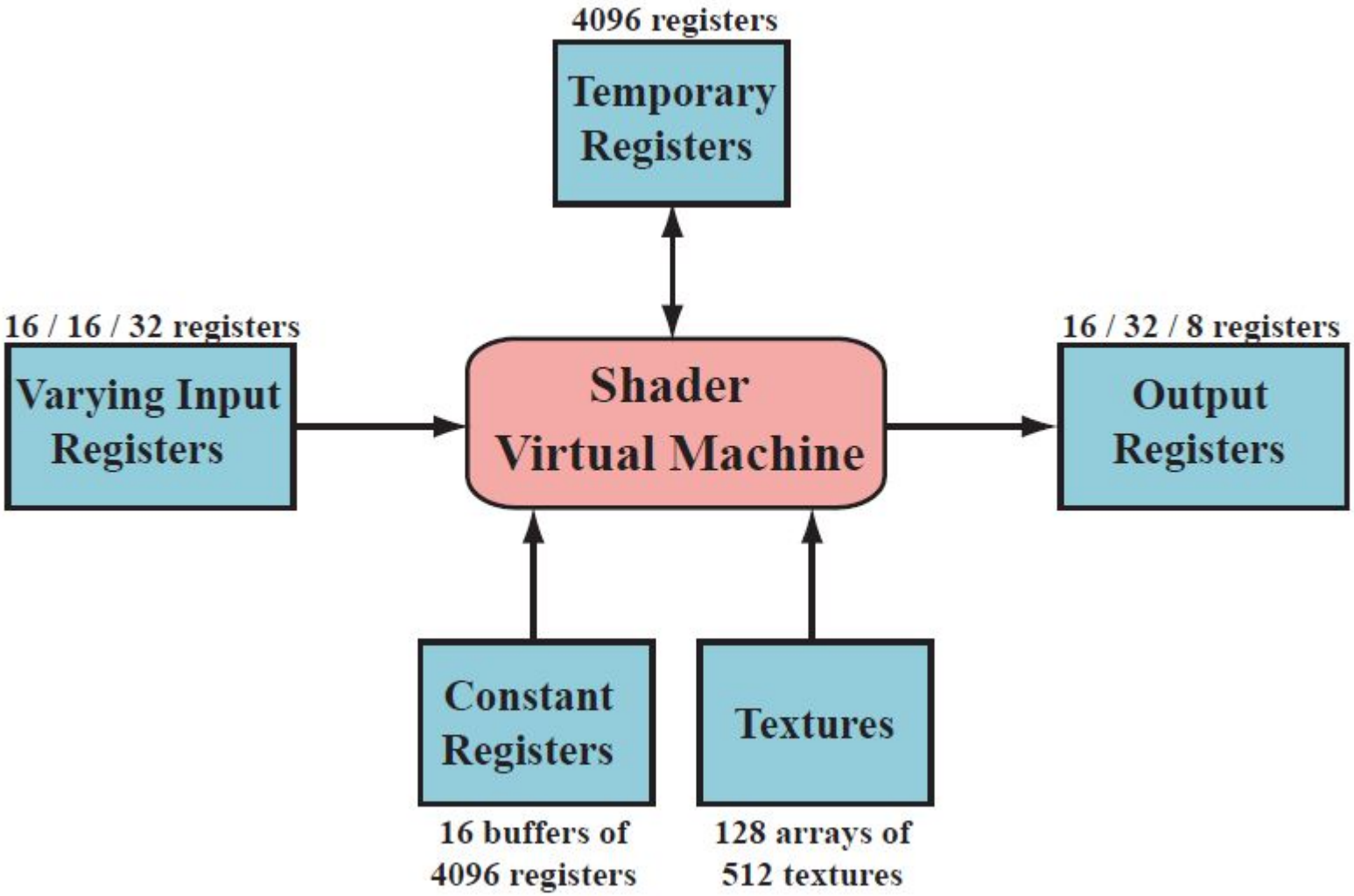
Clip + Screen map



Графический конвейер на GPU



- зеленый – полностью программируемый этап
- желтый – конфигурируемый и непрограммируемый
- синий – полностью фиксированный


4096 registers
Temporary Registers
16 / 16 / 32 registers
Varying Input Registers
Shader Virtual Machine
16 / 32 / 8 registers
Output Registers
Constant Registers
Textures
16 buffers of 4096 registers
128 arrays of 512 textures

Shaders


```
unsigned prg;
. . .
int i;

glUseProgram(prg);
i = glGetUniformLocation(prg, "time");
glUniform1f(i, Time);

glBegin(GL_TRIANGLES);
  glVertex3d(0, 0, 0);
  glVertex3d(1, 0, 0);
  glVertex3d(0, 1, 0);
glEnd();
```

использование

инициализация

```
ShvText = LoadProg("test.glslv");
shv = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(shv, 1, &ShvText, NULL);
glCompileShader(shv);
glGetShaderiv(shv, GL_COMPILE_STATUS, &res);
glGetShaderInfoLog(shv, sizeof(Buf), &len, Buf);

ShfText = LoadProg("test.glslf");
shf = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(shf, 1, &ShfText, NULL);
glCompileShader(shf);
glGetShaderiv(shf, GL_COMPILE_STATUS, &res);
glGetShaderInfoLog(shf, sizeof(Buf), &len, Buf);

prg = glCreateProgram();
glAttachShader(prg, shv);
glAttachShader(prg, shf);

glLinkProgram(prg);
glGetProgramiv(prg, GL_LINK_STATUS, &res);
glGetProgramInfoLog(prg, sizeof(Buf), &len, Buf);
```

Shaders: vertex – закраска Гуро


```
// Gouraud
varying out vec4 c;     uniform vec4 spec;
uniform vec3 eyepos;    uniform vec4 diff;
uniform vec4 amb;       uniform float spower;
uniform float time;

void main( void )
{
  vec3 p = vec3(gl_ModelViewMatrix * gl_Vertex); // transformed point to world space
  vec3 campos = vec3(eyepos);                    // camera position
  vec3 l = vec3(sin(time), 0, cos(time));        // vector to light position
  vec3 v = campos - p;                           // vector to the camera
  vec3 n = gl_NormalMatrix * gl_Normal;          // transformed normal
  gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

  const vec4 diffcolor = vec4(diff);             // diffuse
  const vec4 speccolor = vec4(spec);             // specular
  const vec4 ambcolor = vec4(amb);               // ambient
  const vec4 specpower = vec4(spower);           // specular power

  vec3 n2 = normalize(n);
  vec3 l2 = normalize(l);
  vec3 v2 = normalize(v);
  vec3 r = reflect(-v2, n2);
  vec4 diff = diffcolor * max(dot(n2, l2), 0.0);
  vec4 spec = speccolor * pow(max(dot(l2, r), 0.0), specpower);

  c = diff + spec + ambcolor;
}
```

Shaders: fragment (pixel) – закраска Гуро


```
// Gouraud
varying in vec4 c;

void main( void )
{
  gl_FragColor = c;
}
```

Shaders: vertex – закраска Фонга

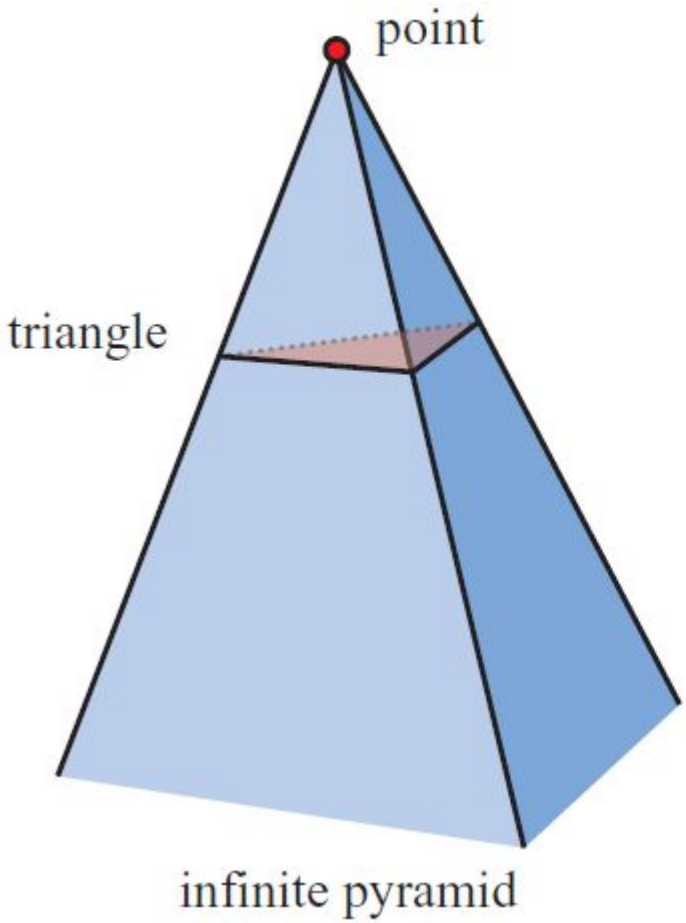

```
// Phong
varying out vec3 l;
varying out vec3 v;
varying out vec3 n;
uniform vec3 eyepos;
uniform float time;

void main( void )
{
  vec3 p = vec3(gl_ModelViewMatrix * gl_Vertex); // transformed point to world space
  vec3 campos = vec3(eyepos);                    // camera position

  l = vec3(sin(time), 0, cos(time));             // vector to light position
  v = campos - p;                                // vector to the camera
  n = gl_NormalMatrix * gl_Normal;               // transformed normal
  gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

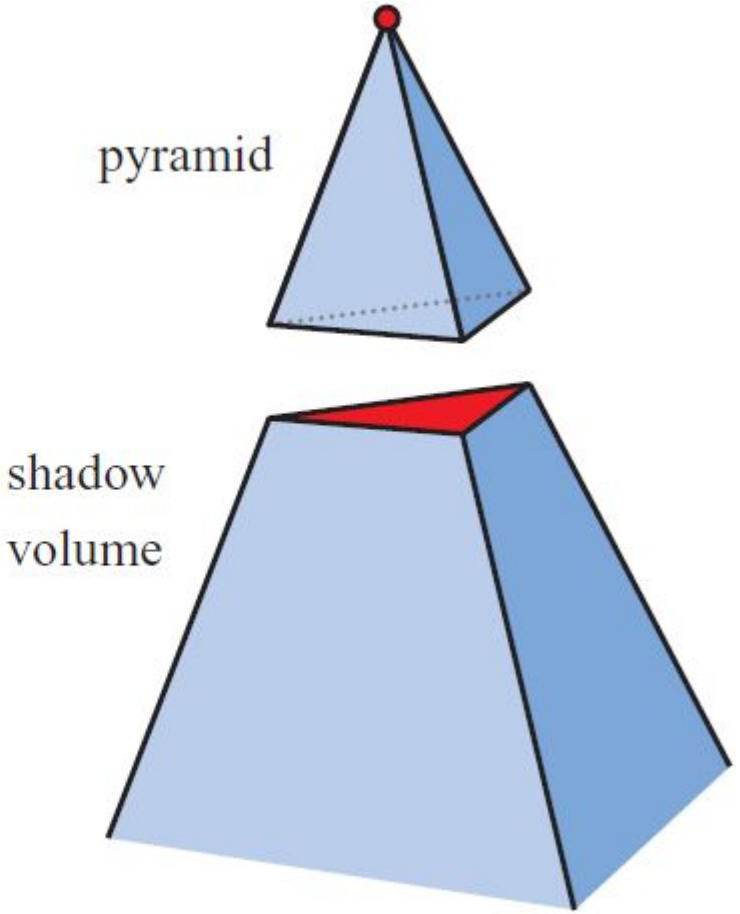
Shaders: fragment (pixel) – закраска Фонга


```
// Phong
varying in vec4 c;

varying in vec3 l;
varying in vec3 v;
varying in vec3 n;
uniform vec4 diff;
uniform vec4 spec;
uniform vec4 amb;
uniform float spower;

void main( void )
{
  const vec4 diffcolor = vec4(diff);   // diffuse
  const vec4 speccolor = vec4(spec);   // specular
  const vec4 ambcolor = vec4(amb);     // ambient
  const vec4 specpower = vec4(spower); // specular power

  vec3 n2 = normalize(n);
  vec3 l2 = normalize(l);
  vec3 v2 = normalize(v);
  vec3 r = reflect(-v2, n2);
  vec4 diff = diff * max(dot(n2, l2), 0.0);
  vec4 spec = spec * pow(max(dot(l2, r), 0.0), spower);

  gl_FragColor = amb + diff + spec;
}
```

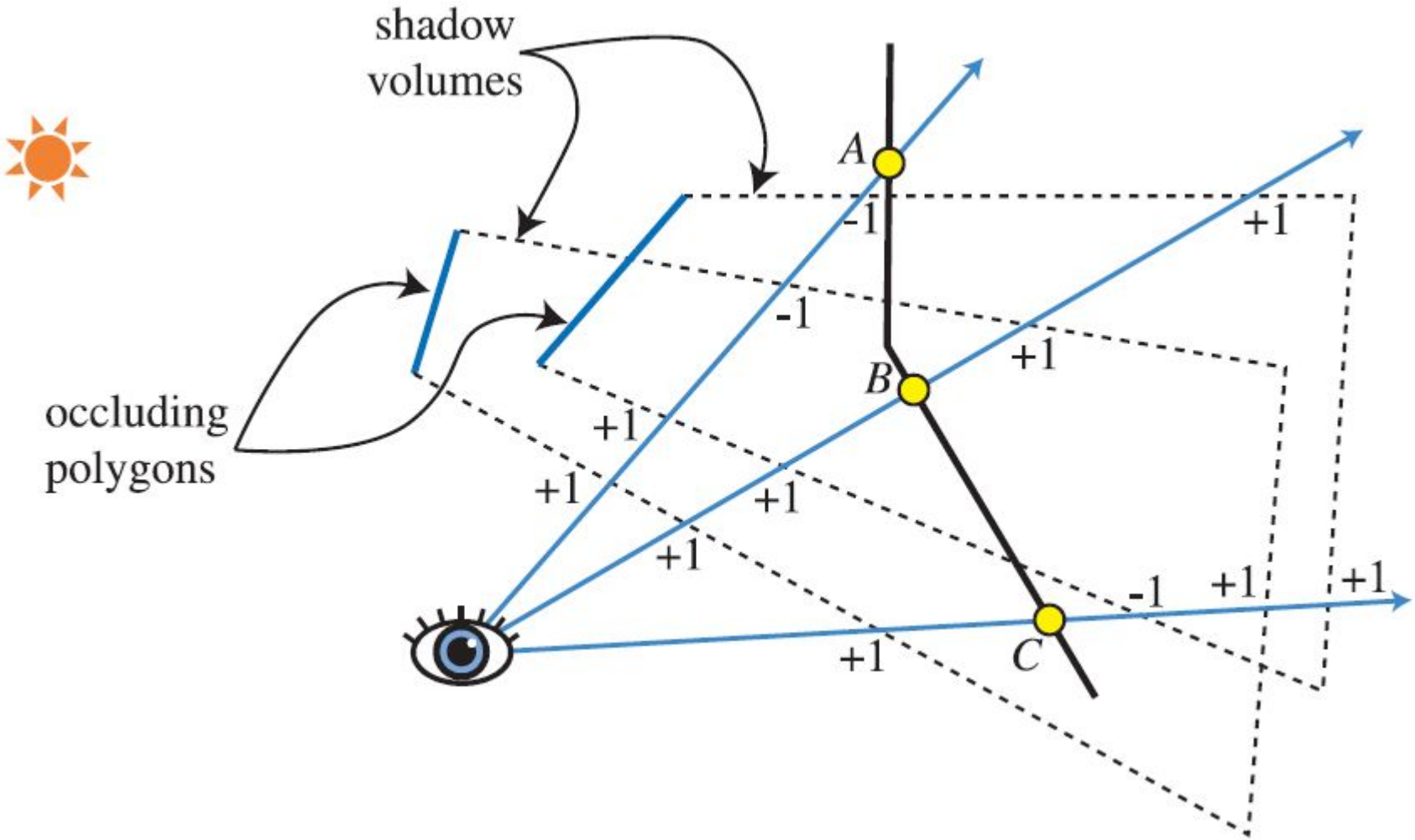
Stencil buffer, blending & alpha test

Stencil buffer: Shadow volume

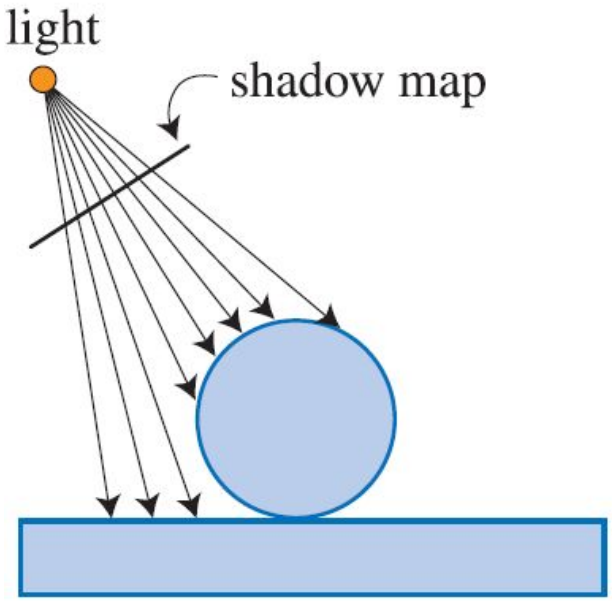


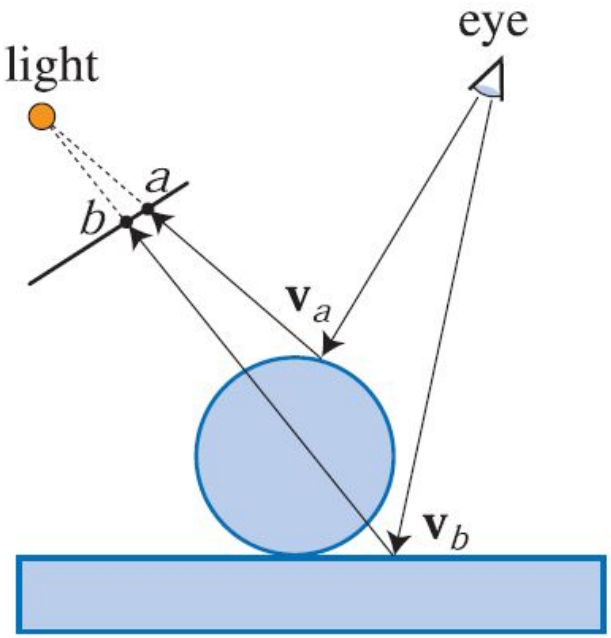


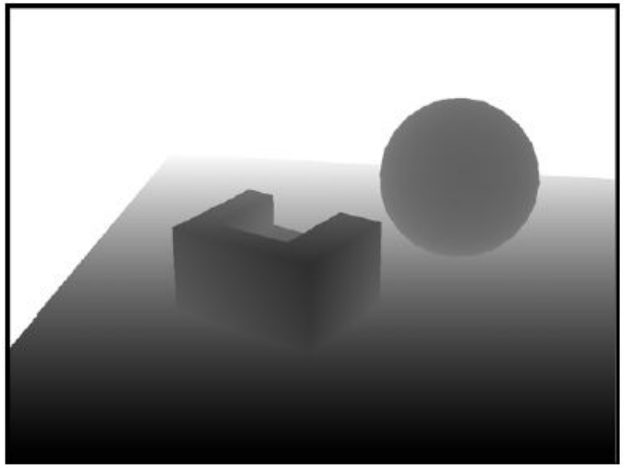
Stencil buffer: Shadow Volume

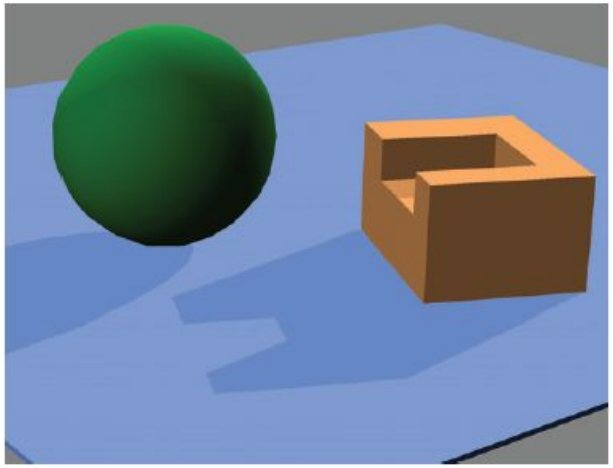


Shadow Map



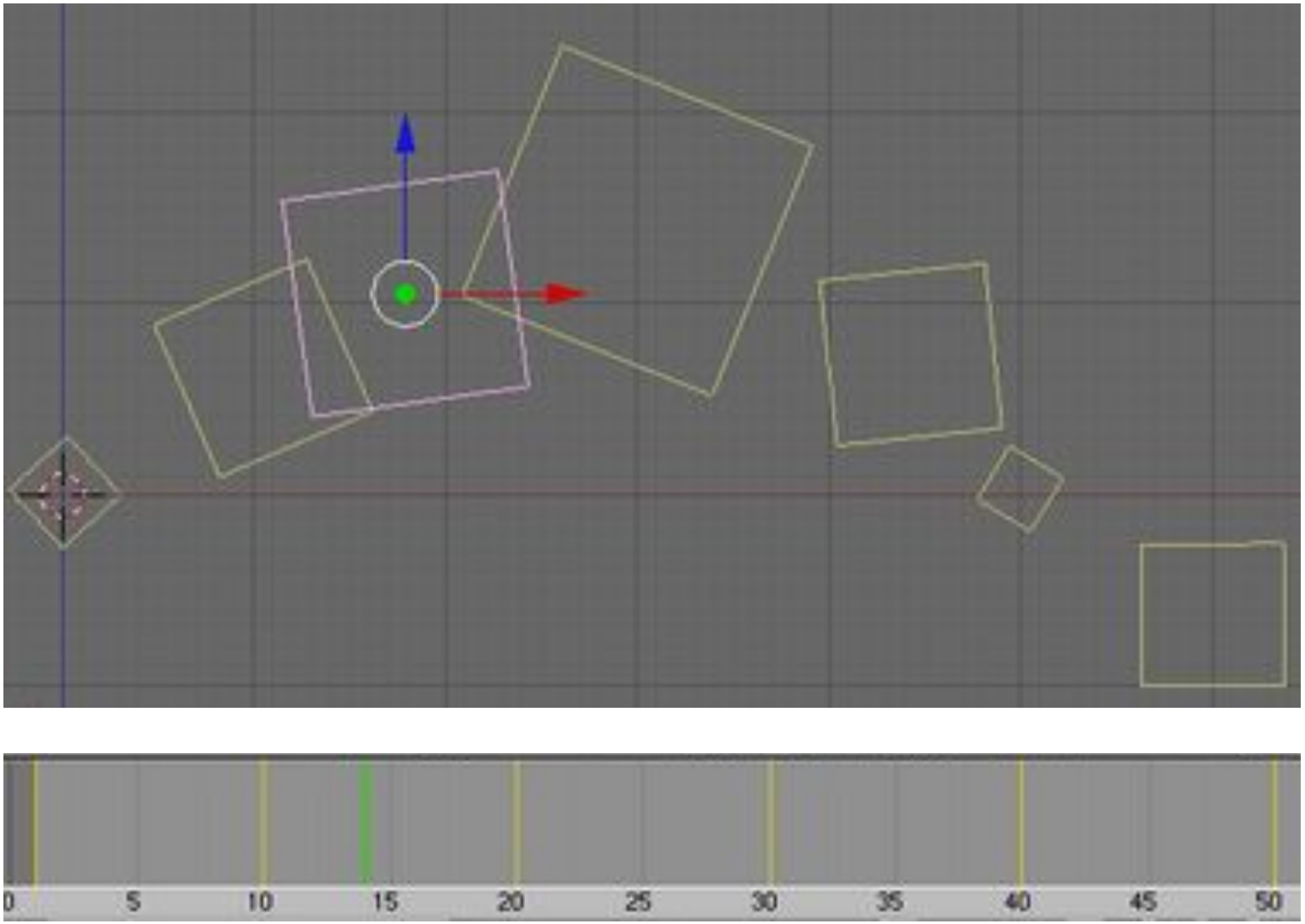




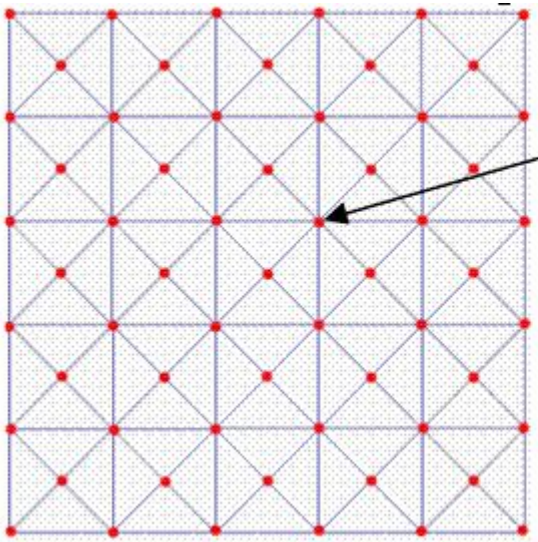


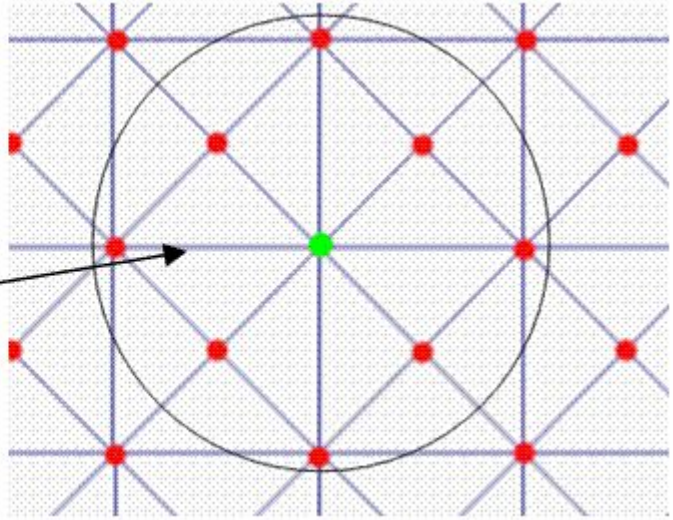
- Синхронизация
  -
```
clock_t start;
. . .
dt = (clock() - start) / (double)CLOCKS_PER_SEC;
```
  -
```
LARGE_INTEGER Start, Quant;
QueryPerformanceFrequency(&Quant);
QueryPerformanceCounter(&Start);
. . .
QueryPerformanceCounter(&Time);
dt = (Time.QuadPart - Start.QuadPart) /
                          (double)Quanty.QuadPart;
```

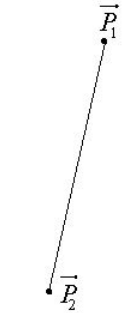
Ключевые кадры (key frame)

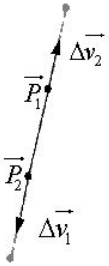


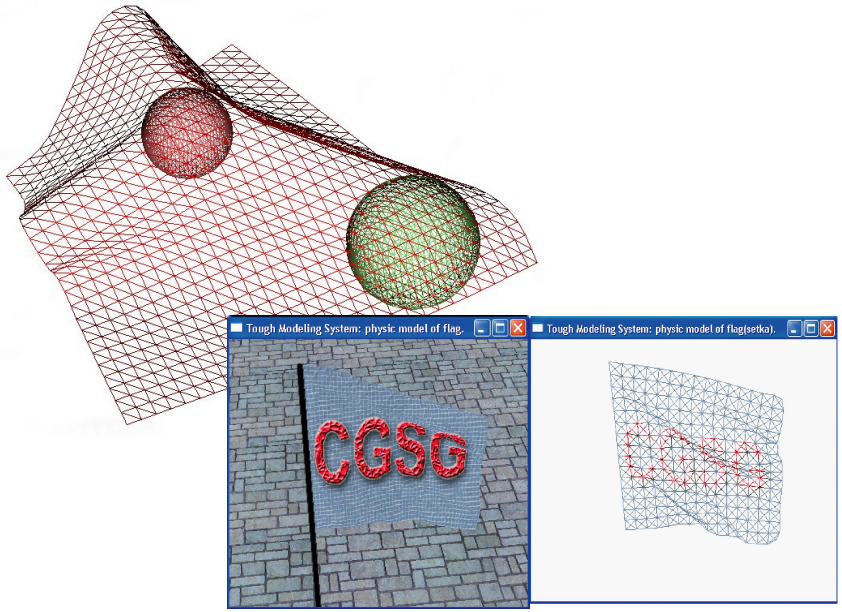
Сеточная анимация



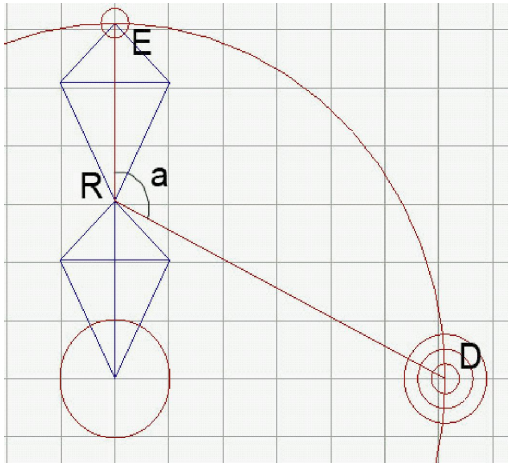


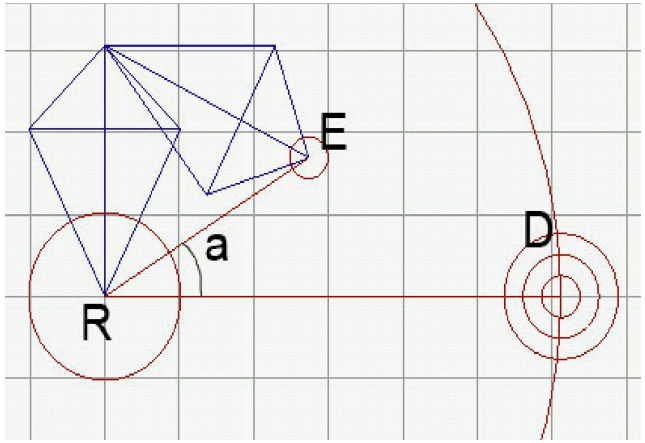


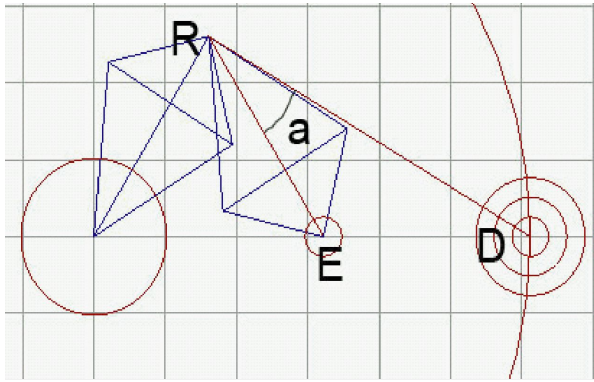


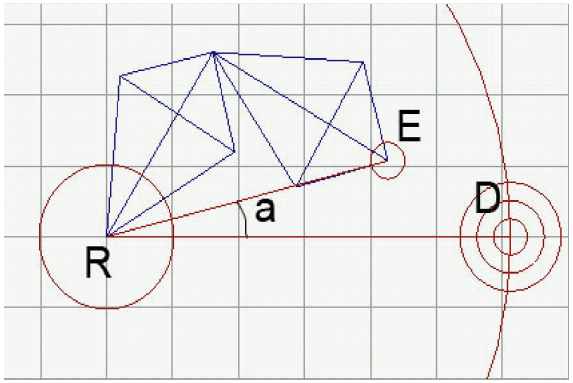


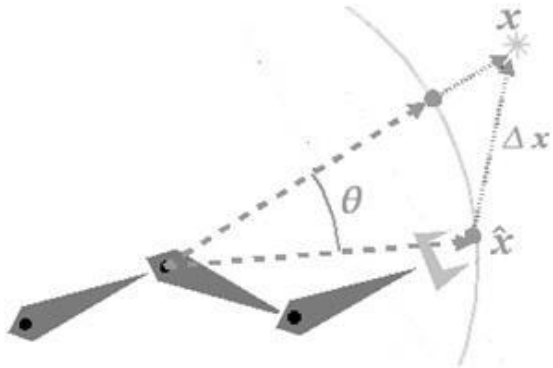
- Cyclic Coordinate Descent (CCD)

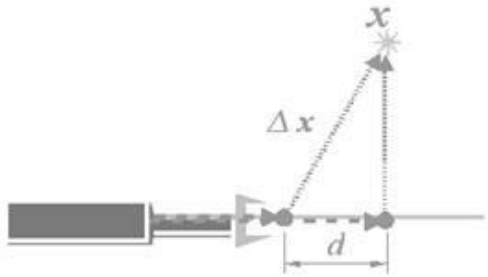


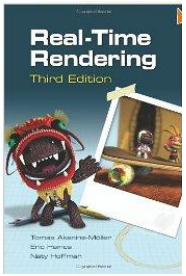




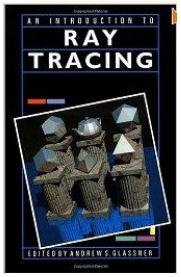




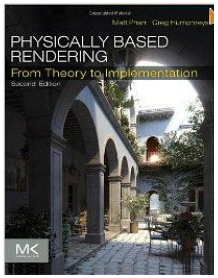




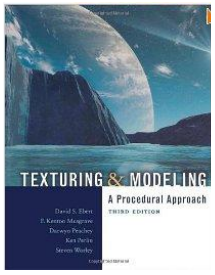
- Tomas Akenine-Moller, Eric Haines, Naty Hoffman, “Real-Time Rendering”, Third Edition, AK Peters-2008



- Andrew S. Glassner (ed.), Eric Haines, Pat Hanrahan, Robert L. Cook, James Arvo, David Kirk, Paul S. Heckbert, "An Introduction to Ray Tracing (The Morgan Kaufmann Series in Computer Graphics)", Academic Press-1989.



- Matt Pharr, Greg Humphreys, "Physically Based Rendering, Second Edition: From Theory To Implementation", Morgan Kaufmann-2010



- David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steve Worley, "Texturing and Modeling, Third Edition: A Procedural Approach (The Morgan Kaufmann Series in Computer Graphics)”, Morgan Kaufmann-2002

- Практические задания (до зачетного занятия)
  - Реализовать простейшую анимацию сферы с использованием библиотеки OpenGL или Direct3D (со встроенными в библиотеку возможностями фиксированного конвейера, например, освещения)
  - К предыдущему заданию добавить освещение, реализованное на шейдерах (предпочтительно на фрагментных/пиксельных для поточечного освещения).

В обоих заданиях не использовать retained библиотек (фигуру построить с использованием базовых примитивов).