

OpenGL

- a quick guide

Ulf Assarsson
Department of Computer Engineering
Chalmers University of Technology



OpenGL vs Direct3D

- Direct3D

- Microsoft, Sept. '95 on Windows95
- Common for games
- Historically: “Adapted to graphics hardware evolution”
 - Now: influences hardware features perhaps more than OpenGL
- (Now after many upgrades very similar to OpenGL)

**Direct3D was
messy to
program version
3.0 – 6.0.
Today version 11**

- OpenGL

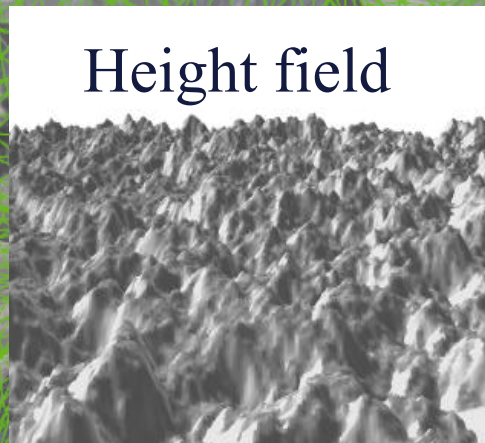
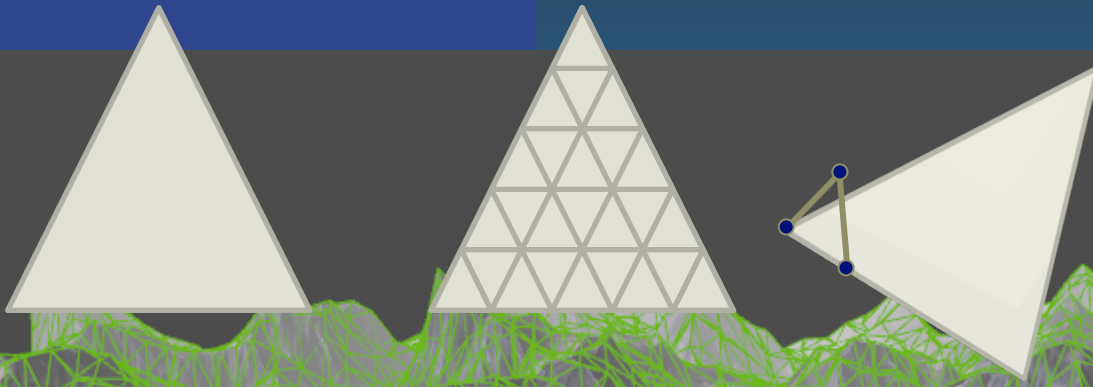
- SGI
- Historically: “Precede the hardware evolution”
- Operation system independent
- Window system independent
- Industry, games (Quake – thanks John Carmack)
- January 1992
- Extendable, stable, better design,

**Probably why
OpenGL still is
largely used.**

OpenGL Evolution

- Controlled by an Architectural Review Board (ARB)
 - Members include ~~SGI~~, Microsoft, Nvidia, AMD, HP, S3, IBM,.....
 - Present version 4.3
 - Evolution reflects new hardware capabilities
 - **More functionality for vertex / fragment programs**
 - **Geometry shaders,**
 - **Tessellation units**
 - DX11: Hull shader = GL: Tessellation Control Shader
 - Domain shader = Tessellation Evaluation Shader
 - Allows for platform specific features through extensions

Tessellation – brief glance



Input Assembler

Vertex Shader

Hull Shader

Tessellator

Domain Shader

Geometry Shader

Rasterizer

Pixel Shader

Output Merger

Overview of today's OpenGL lecture

- OpenGL
 - Specifying vertices and polygons, Buffer Objects
 - Shaders
 - Framebuffer Objects
 - Texturing
 - Shadow Maps!
 - Blending
 - Buffers (frame b/f/l/r, depth, alpha-channel, stencil)
 - Misc: point/line width, clip planes
- GLU – The OpenGL Graphics System Utility Library
- GLUT – The OpenGL Utility Toolkit
 - Windows and menus
 - Callbacks for events
 - Text support
 - Predefined Objects

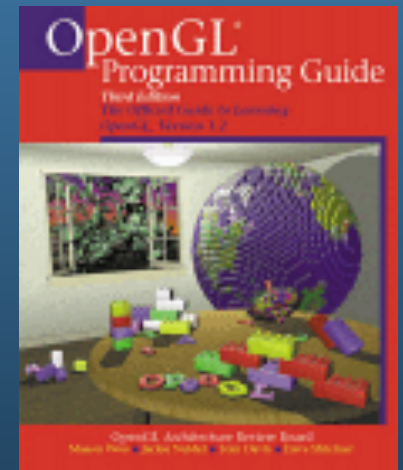
OpenGL – links

- <http://www.khronos.org/files/opengl43-quick-reference-card.pdf>
- Home page: www.opengl.org
- Sample code: http://www.opengl.org/wiki/Code_Resources/
- OpenGL 4.3 specification:
 - <http://www.opengl.org/registry/doc/GLSLangSpec.4.30.6.pdf>
- GLU specification: <http://www.cse.chalmers.se/~uffe/glu1.3.pdf>
- GLUT specification:
<http://www.cse.chalmers.se/~uffe/glut-3.spec.pdf>

ALSO ON COURSE HOME PAGE:

<http://www.cse.chalmers.se/edu/course/TDA361/>

- Programmers Manual and Reference Manual:
 - <http://www.cse.chalmers.se/edu/course/TDA361/redbook.pdf>
 - BUT IT IS HEAVILY OUTDATED BY NOW.
 - You can buy the latest RedBook 8:th Ed. for OpenGL 4.1

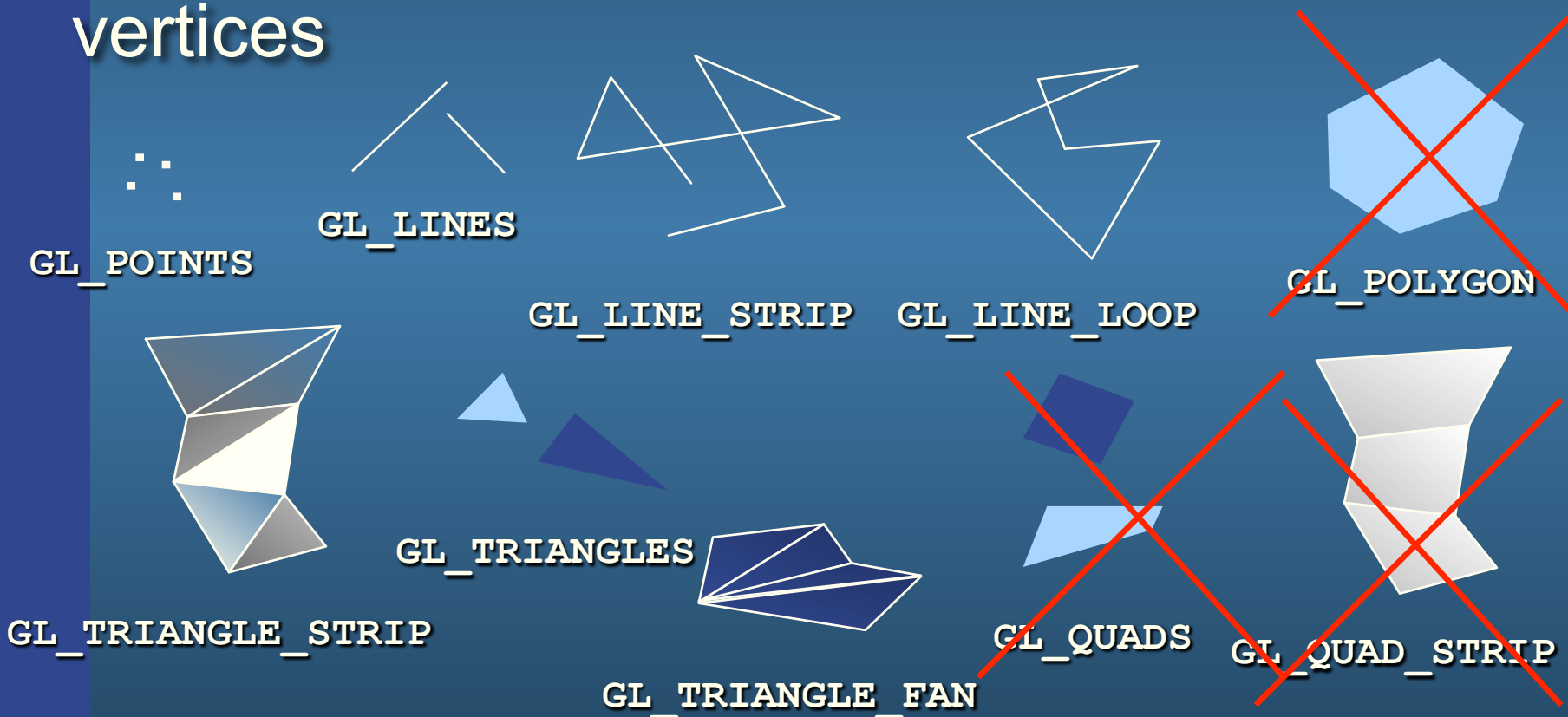


Include

- `#include <GL/gl.h>`
- Links with `OpenGL32.lib` (MS Windows)
- `glew.h` / `glew32.lib` / `glew32.dll`
- (`GLee.h` / `GLee.cpp`)

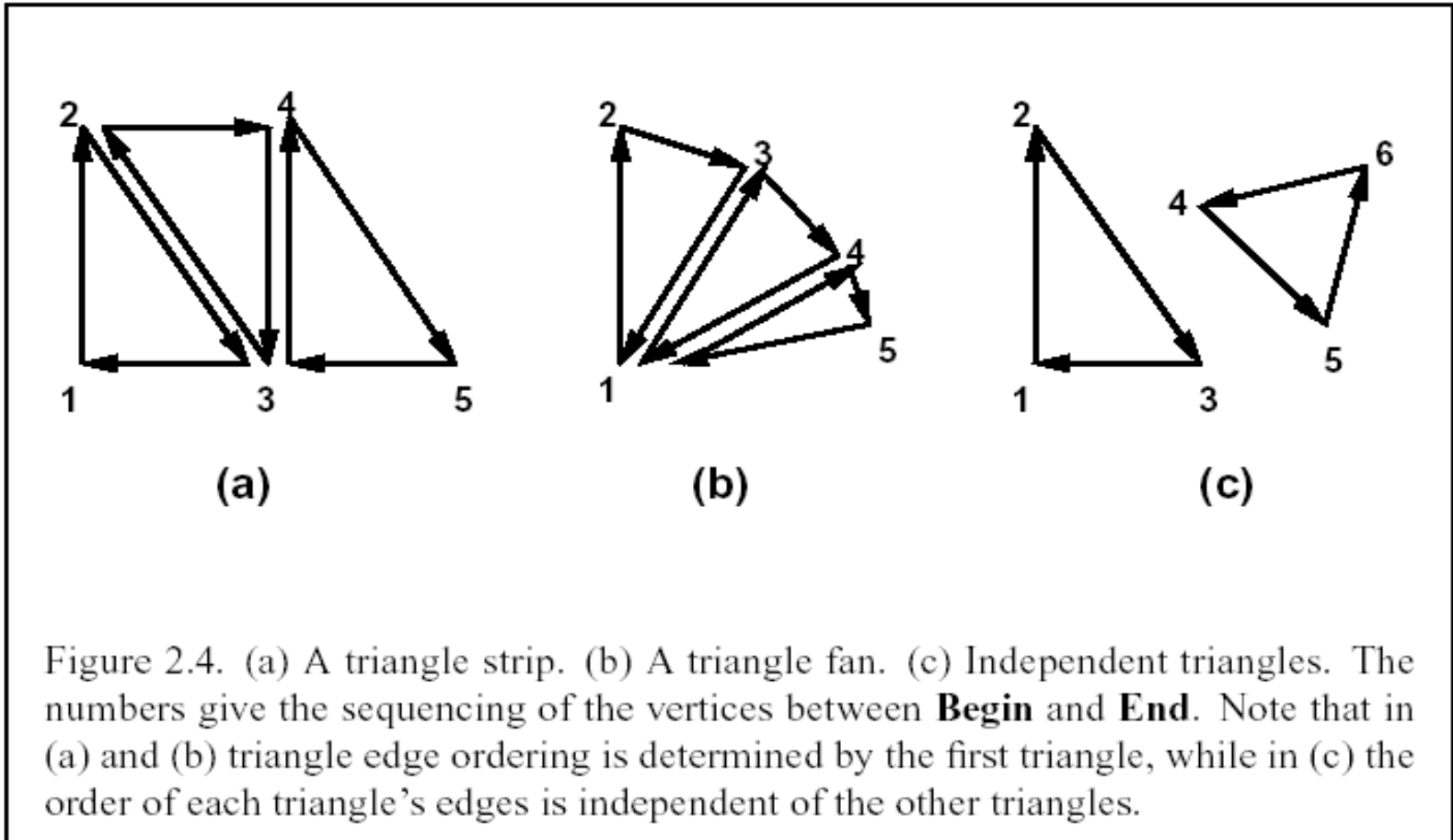
OpenGL Geometric Primitives

- All geometric primitives are specified by vertices



Vertex order

`glFrontFace(enum dir) CCW, CW`
`CullFace(enum mode) -- mode: FRONT, BACK, FRONT_AND_BACK`
`glEnable/Disable(CULL_FACE)`



Note: Vertex order indicates that all but the last triangle is backfacing with CCW-ordering (default for OpenGL).

Specifying vertices and polygons

- OpenGL is a state machine. Commands typically change the current state

Historical Commands:

- Multiple calling formats for the commands: `void glVertex{234}{sfd}(T coords);`
- `glBegin()/glEnd().` (Slow)

```
glBegin(GL_TRIANGLE)
  glVertex3f(0,0,0)
  glVertex3f(0,1,0);
  glVertex3f(1,1,0);
glEnd();
```

Optional: Can specify for instance `glColor3f(r,g,b)`, `glTexCoord2f(s,t)`, `glNormal3f(x,y,z)` - typically per vertex or per primitive.

TODAY, WE RATHER USE VERTEX ARRAYS

- Vertex Arrays (Fast):

```
void DrawArrays(enum mode, int first, sizei count);
```

```
void MultiDrawArrays(enum mode, int *first, sizei *count, sizei primcount);
```

```
void DrawElements(enum mode, sizei count, enum type, void *indices);
```

Using index list

other options exist - see the OpenGL Reference Manual online

MultiDrawArrays:

```
for (i = 0; i < primcount; i++)
```

```
  DrawArrays(mode, first[i], count[i]);
```

Example of using Vertex Arrays

1. // SEND THE VERTEX COORDINATES TO A BUFFER

```
glGenBuffers( 1, &coordBuffer );           // Create a handle for the coordinate buffer
glBindBuffer( GL_ARRAY_BUFFER, coordBuffer ); // Set the newly created buffer as the current one
glBufferData( GL_ARRAY_BUFFER, sizeof(coords), coords, GL_STATIC_DRAW ); // Send the data
```

// Do the same thing for the color data

```
glGenBuffers( 1, &colorBuffer );
glBindBuffer( GL_ARRAY_BUFFER, colorBuffer );
glBufferData( GL_ARRAY_BUFFER, sizeof(colors), colors, GL_STATIC_DRAW );
```

```
Float coords[] = {
// X   Y   Z
  0.0f, 0.5f, 1.0f, // v0
 -0.5f, -0.5f, 1.0f, // v1
  0.5f, -0.5f, 1.0f, // v2
  0.0f, -1.0f, 1.0f  // v3
};
```

```
float colors[] = {
// R   G   B
  1.0f, 0.0f, 0.0f, // Red
  0.0f, 1.0f, 0.0f, // Green
  0.0f, 0.0f, 1.0f, // Blue
  1.0f, 1.0f, 0.0f  // Yellow
}
```

// Connect triangle data with a Vertex Array Object and the Vertex shader

```
glGenVertexArrays(1, &vertexArrayObject);
glBindVertexArray(vertexArrayObject);
```

// Connects coordBuffer to vertexArrayObject and activates coordBuffer for next command below.

```
glBindBuffer( GL_ARRAY_BUFFER_ARB, coordBuffer );
glVertexAttribPointer(0, 3, GL_FLOAT, false/*normalized*/, 0/*stride*/, 0/*offset*/);
```

// Connects colorBuffer to vertexArrayObject and activates colorBuffer for command below.

```
glBindBufferARB( GL_ARRAY_BUFFER_ARB, colorBuffer );
glVertexAttribPointer(1, 3, GL_FLOAT, false/*normalized*/, 0/*stride*/, 0/*offset*/);
```

```
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
```

```
in vec3 vertex;           VERTEX SHADER
in vec3 color;
out vec3 outColor;
uniform mat4 modelViewProjectionMtx;

void main() {
  gl_Position = modelViewProjectionMtx*
                vec4(vertex,1);
  outColor = color;
}
```

2. // Just before linking the shader program, you should specify:

```
glBindAttribLocation(shaderProgram, 0, "vertex");
glBindAttribLocation(shaderProgram, 1, "color");
```

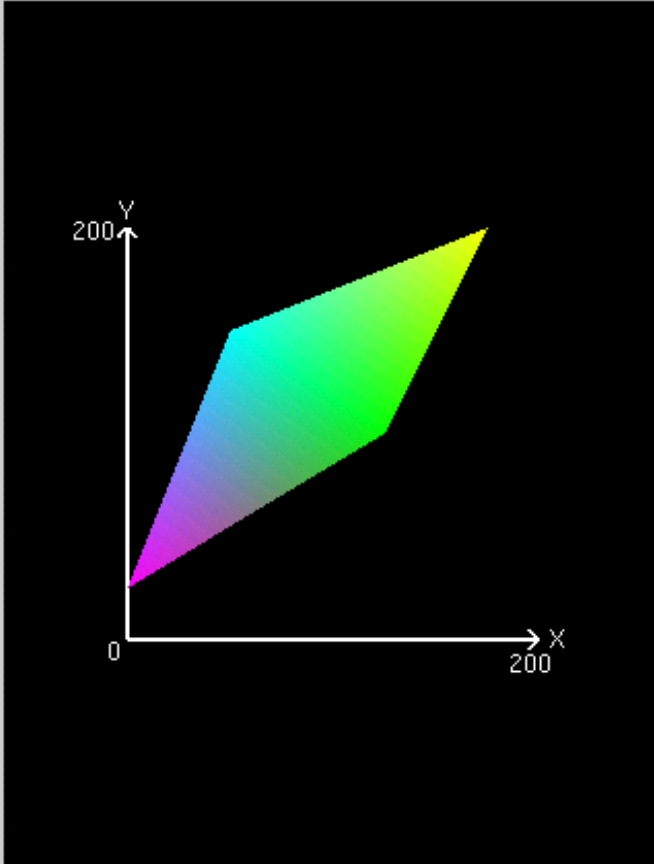
3. COMMANDS TO DRAW

```
glUseProgram( shaderProgram );
glBindVertexArray(vertexArrayObject);
glDrawArrays( GL_TRIANGLE_STRIP, 0, 4 );
```

Example of historical slow way:

Shapes

Screen-space view



Command manipulation window

```
glBegin (GL_TRIANGLE_STRIP);  
glColor3f (1.00 , 0.00 , 1.00 );  
glVertex2f (0.0 , 25.0 );  
glColor3f (0.00 , 1.00 , 1.00 );  
glVertex2f (50.0 , 150.0 );  
glColor3f (0.00 , 1.00 , 0.00 );  
glVertex2f (125.0 , 100.0 );  
glColor3f (1.00 , 1.00 , 0.00 );  
glVertex2f (175.0 , 200.0 );  
glEnd();
```

Example of a GfxObject Class

```
class GfxObject {  
public:
```

```
    Object() {};  
    ~Object() {};  
    render(); E.g.:  
    load("filename");
```

...

```
private:
```

```
    Matrix4x4  
    std::vector<vec3f>  
    std::vector<vec3f>  
    std::vector<vec2f>  
    std::vector<vec3f>
```

```
or just:
```

```
    GLhandle  
    GLuint
```

```
}
```

```
{  
    glUseProgram(m_shaderProgram);  
    int loc = glGetUniformLocation(shaderProgram,  
        "modelViewProjectionMatrix");  
    glUniformMatrix4fv(loc, 1, false,  
        &modelViewProjectionMatrix);  
  
    glBindVertexArray(m_vertexArrayObject);  
    glDrawArrays( GL_TRIANGLES, 0, m_vertices.size());  
}
```

```
    m_modelToWorldTransform;  
    m_vertices;  
    m_normals;  
    m_texCoords;  
    m_colors;
```

```
    m_shaderProgram;  
    m_vertexArrayObject;
```

Triangle data is necessary for collision detection and updating of data.

Texture Mapping

You probably recognize from lab 2

- Three steps

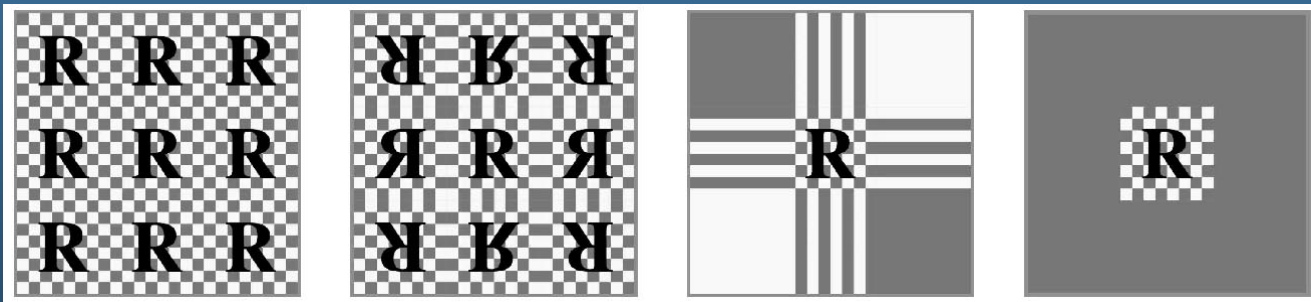
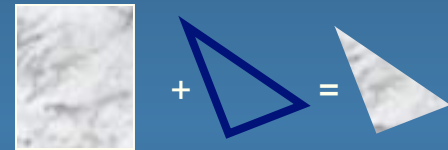
- ① specify texture

- read or generate image
 - assign to texture – `glGenTextures()`, `glBindTexture()`, `glTexImage2D()`, `glGenerateMipMap()`

- ② assign texture coordinates to vertices

- ③ specify texture parameters

- set texture filter – `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, ...)`
 - set texture wrap mode – `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, ...)`



Texture Mapping

Specifying Texture:

```
texID = ilutGLLoadImage("flake.ppm"); // Here, we use DevIL
```

```
glActiveTexture(enum texUnit) -- specify texture unit (up to 32)
```

```
glBindTexture(texID), -- specify texture ID that this texture unit and data is identified with
```

```
glTexImage1/2/3D (), glCopyTexSubImage2D() -- set / affect image data
```

```
glGenerateMipMap() -- Create the mipmap hierarchy
```

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY, ...)
```

```
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)
```

```
in vec3 vertex;          VERTEX SHADER
in vec3 texCoordIn;
out vec3 texCoord;
uniform mat4 modelViewProjectionMtx;
void main() {
    gl_Position = modelViewProjectionMtx *
                vec4(vertex,1);
    texCoord = texCoordIn;
}
```

Specifying Texture Coordinates

1. // Send the TEXTURE COORDINATES to a buffer

```
glGenBuffers( 1, &texcoordBuffer ); // Create a handle for the texcoord buffer
glBindBuffer( GL_ARRAY_BUFFER, texcoordBuffer ); // Set the newly created buffer as the current one
glBufferData( GL_ARRAY_BUFFER, sizeof(texcoords), texcoords, GL_STATIC_DRAW ); // Send the data
```

```
float texcoords[] = {
    0.0f, 1.0f,
    0.0f, 0.0f,
    1.0f, 0.0f,
    1.0f, 1.0f
};
```

// Connect texcoord data with the Vertex Array Object and the Vertex shader

```
glBindVertexArray(vertexArrayObject);
```

```
// Connects texcoordBuffer to vertexArrayObject
```

```
glBindBuffer( GL_ARRAY_BUFFER_ARB, texcoordBuffer );
glVertexAttribPointer(1, 2, GL_FLOAT, false/*normalized*/, 0/*stride*/, 0/*offset*/);
```

```
glEnableVertexAttribArray(2);
```

FRAGMENT SHADER

```
uniform sampler2D tex0;

in vec2 texCoord;

void main()
{
    gl_FragColor = texture2D(tex0,
                            texCoord.xy);
}
```

```
useProgram (shaderProgram) ...
```

```
int texLoc = glGetUniformLocationARB( shaderProgram, "tex0" );
```

```
glUniform1iARB( texLoc, 0 );
```


Example of Loading a Texture

Do once when loading texture:

```
texture = ilutGLLoadImage("flake.ppm");           // Here, we use DevIL
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture);
glGenerateMipmap(GL_TEXTURE_2D);

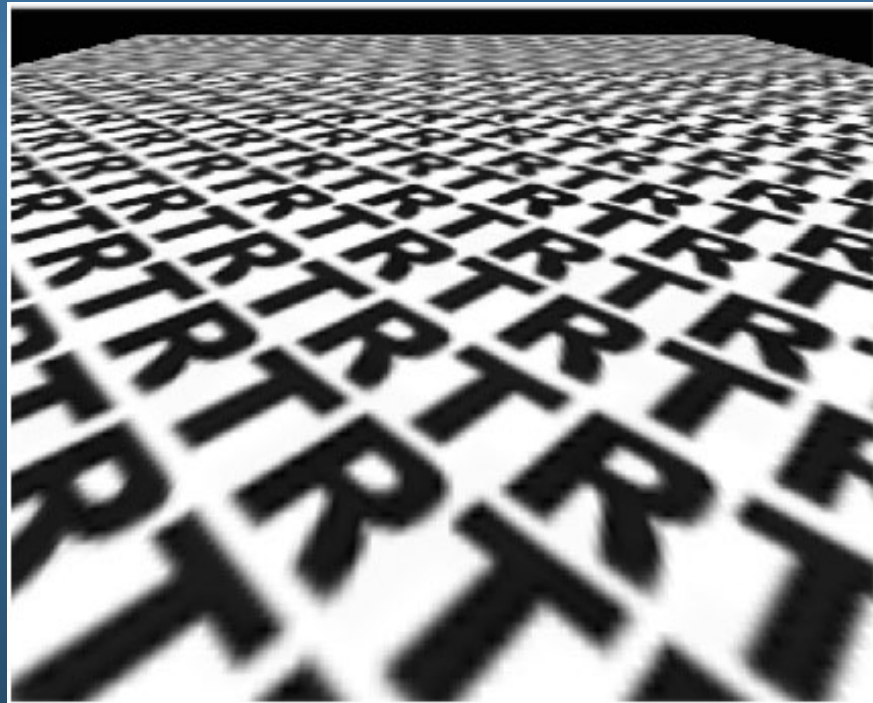
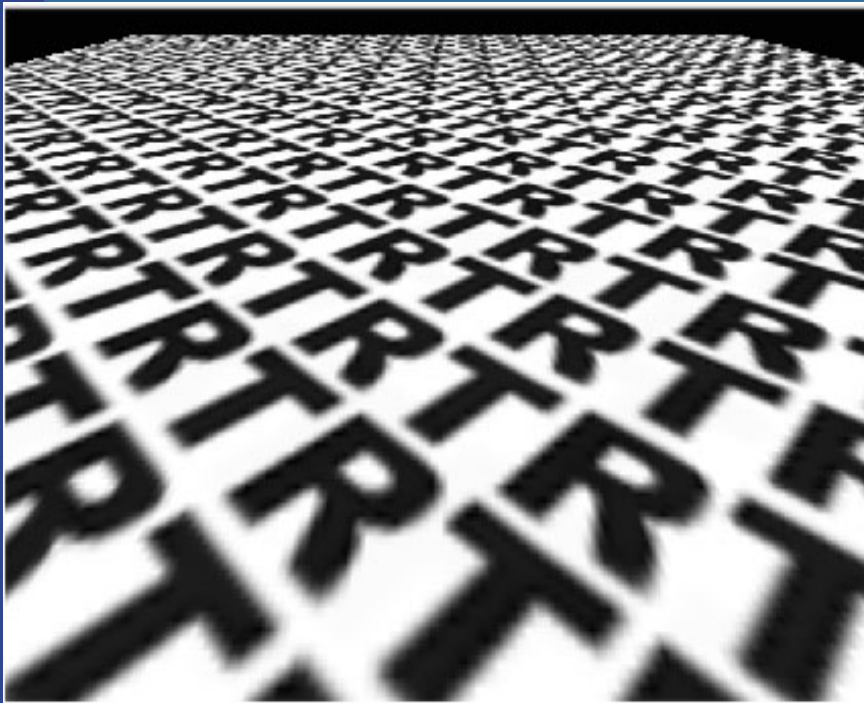
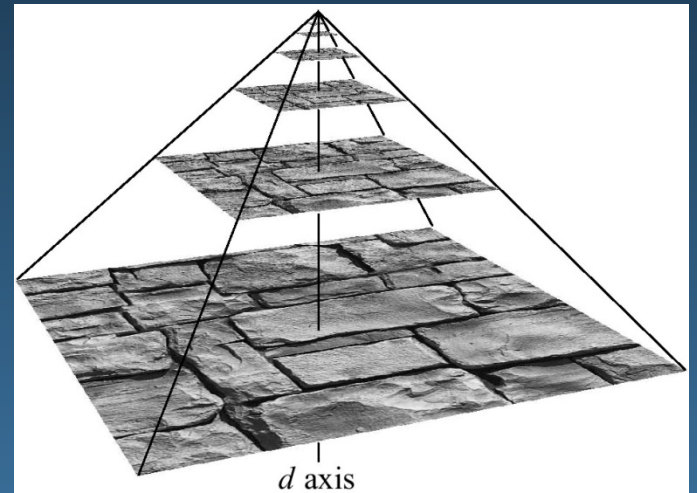
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, 16);

//Indicates that the active texture should be repeated over the surface
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// Sets the type of mipmap interpolation to be used on magnifying and
// minifying the active texture. These are the nicest available options.
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
```

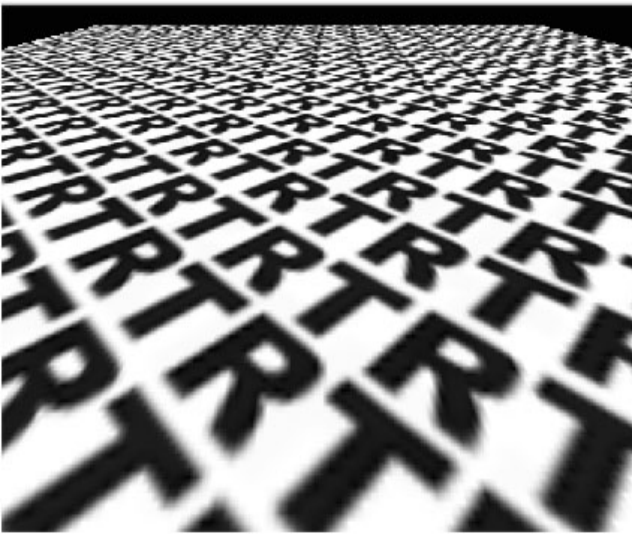
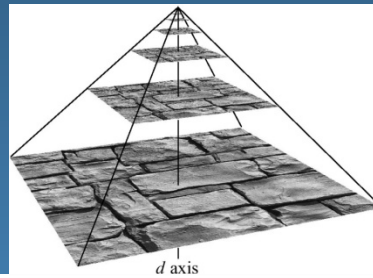
Do every time you want to use this texture when drawing:

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture);
```

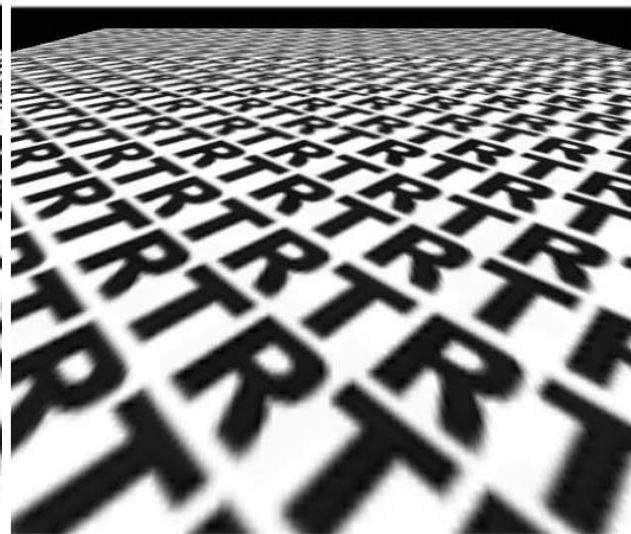
Mip Mapping



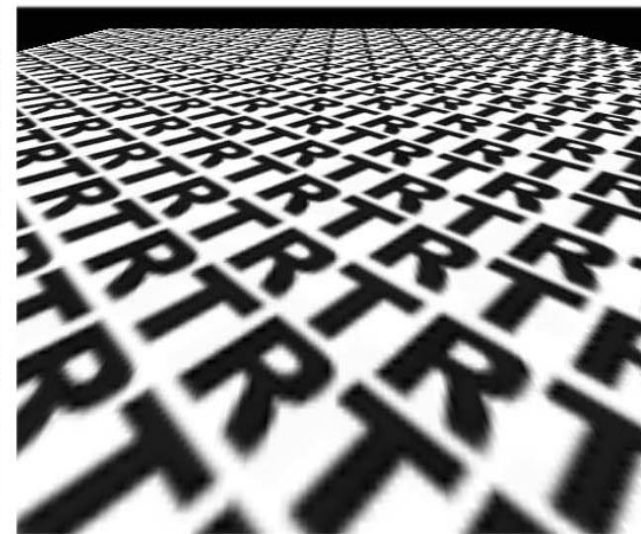
Anisotropic filtering



No filtering



Mipmapping



Anisotropic

Anisotropic filtering and auto-mipmap generation

Enabling anisotropic filtering:

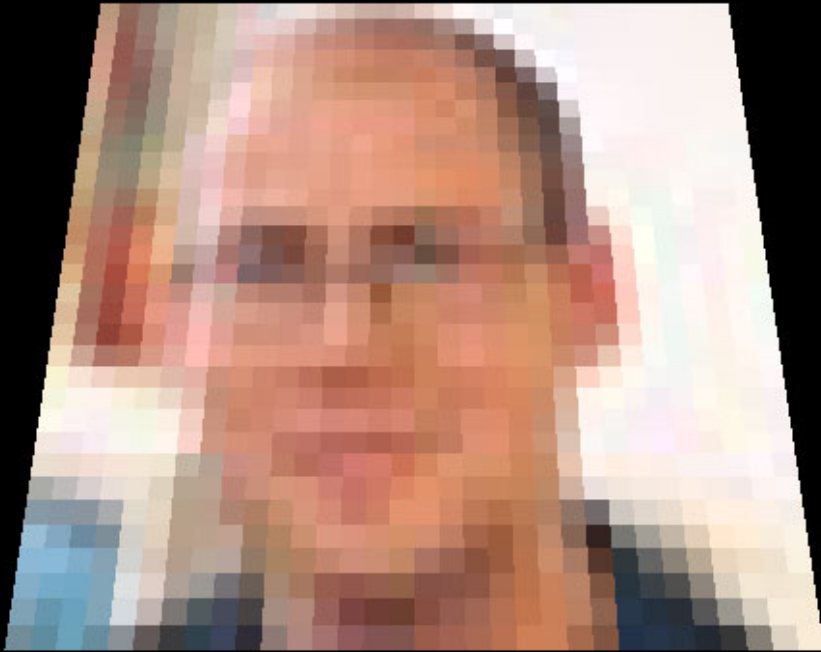
- float MaxAnisotropy
- glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, &MaxAnisotropy);

- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, MaxAnisotropy);

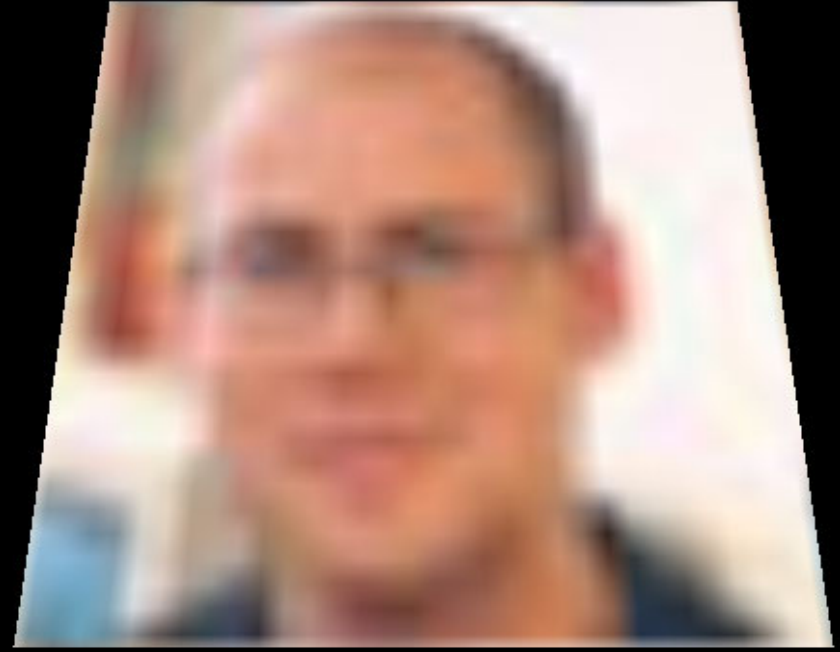
~~Enabling autogeneration of mipmaps (mipmaps are recomputed when the texture data changes) :~~

- ~~• glTexParameteri(GL_TEXTURE_2D, GL_GENERATE_MIPMAP_SGIS, GL_TRUE);~~

Examples of filtering



Nearest



Linear

Specifying a Texture: Other Methods

- Use frame buffer as source of texture image
 - uses current buffer as source image

`glCopyTexImage1D(...)`

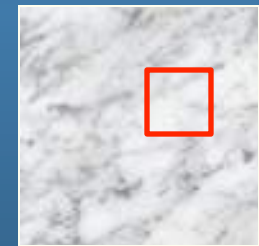
`glCopyTexImage2D(...)`

- Modify part of a defined texture

`glTexSubImage1D(...)`

`glTexSubImage2D(...)`

`glTexSubImage3D(...)`



- Do both with `glCopyTexSubImage2D(...)`, etc.

Reflections with environment mapping

- Texture lookups from an environment map



VERTEX SHADER

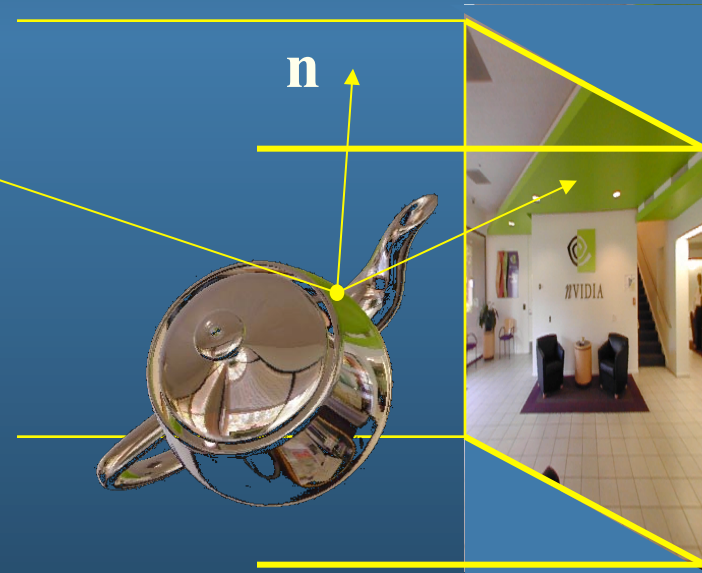
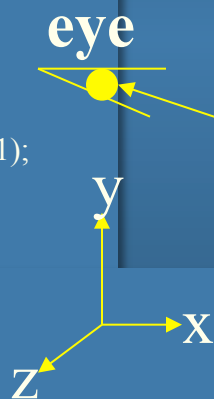
```
in vec3    vertex;  
in vec3    normalIn;    // The normal  
out vec3    normal;  
out vec3    eyeVector;  
uniform mat4 normalMatrix;  
uniform mat4 modelViewMatrix;  
uniform mat4 modelViewProjectionMatrix;
```

```
void main()  
{  
    gl_Position = modelViewProjectionMatrix * vec4(vertex,1);  
    eyeVector = (modelViewMatrix * vec4(vertex, 1)).xyz;  
    normal = (normalMatrix * vec4(normalIn,0.0)).xyz;  
}
```

FRAGMENT SHADER

```
in vec3 normal;  
in vec3 eyeVector;  
uniform samplerCube tex0;  
out vec4 fragmentColor;
```

```
void main()  
{  
    vec3 reflectionVector = normalize(reflect(normalize(eyeVector),  
    normalize(normal)));  
    fragmentColor = texture(tex0, reflectionVector);  
}
```



Shadow Maps



Shadow Maps

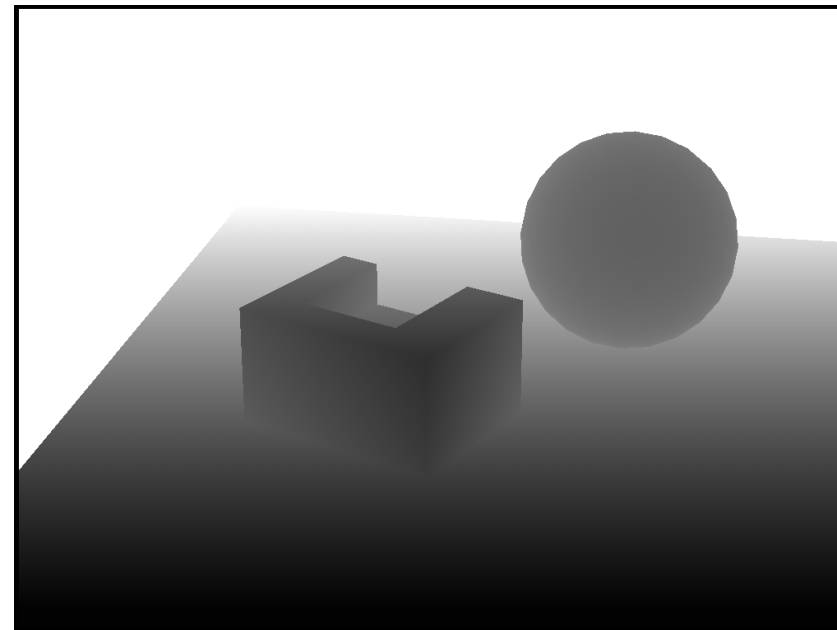
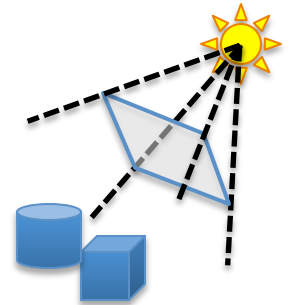


Shadow Maps

Basic Algorithm – the simple explanation:

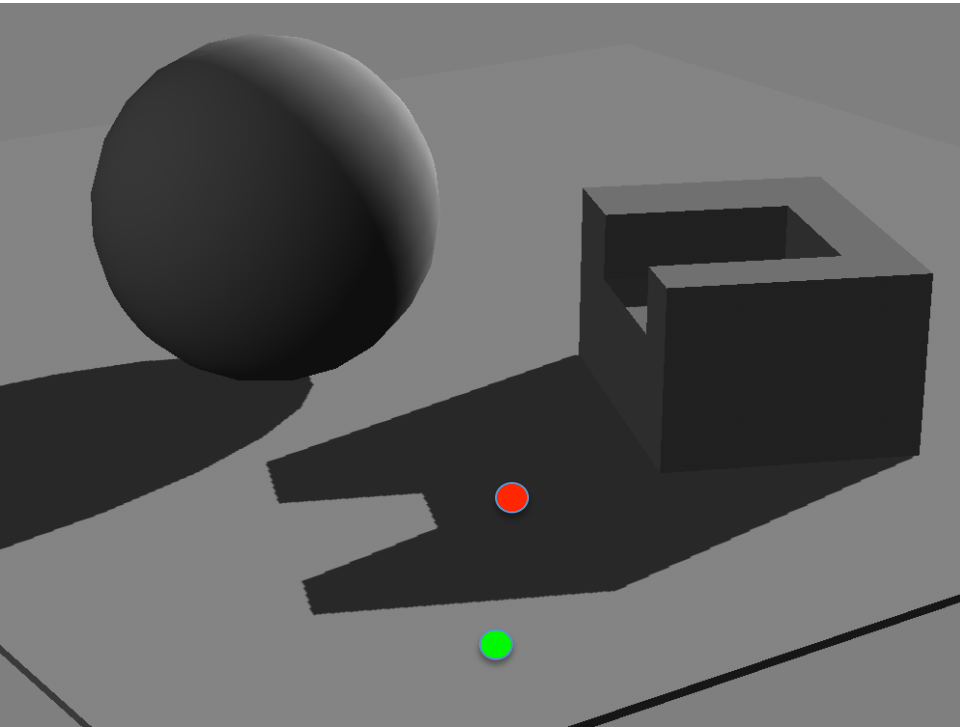
Idea:

- Render image from light source
 - Represents geometry in light
- Render from camera
 - Test if rendered point is visible in the light's view
 - If so -> point in light
 - Else -> point in shadow



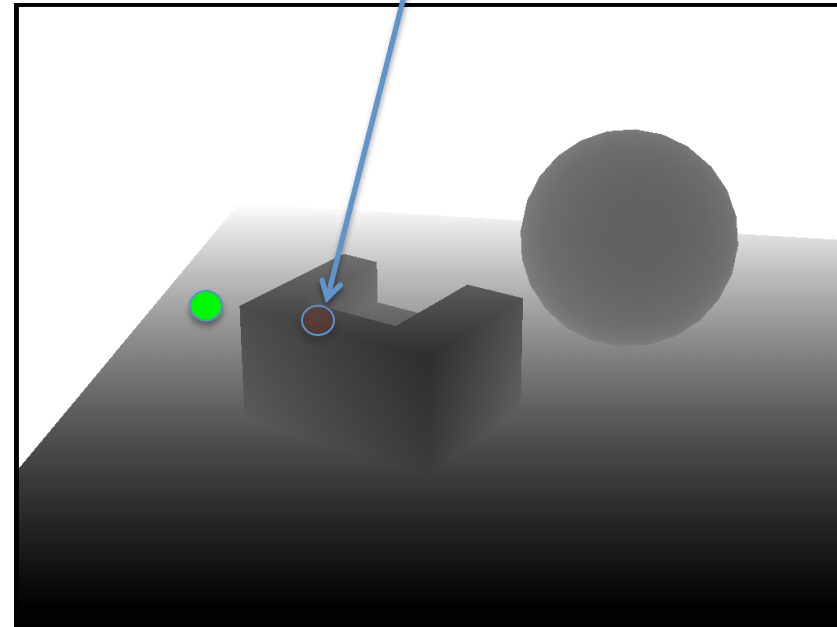
Shadow Map (light's view)

Shadow Maps



Camera's view

Point not represented in shadow map (point is behind box)

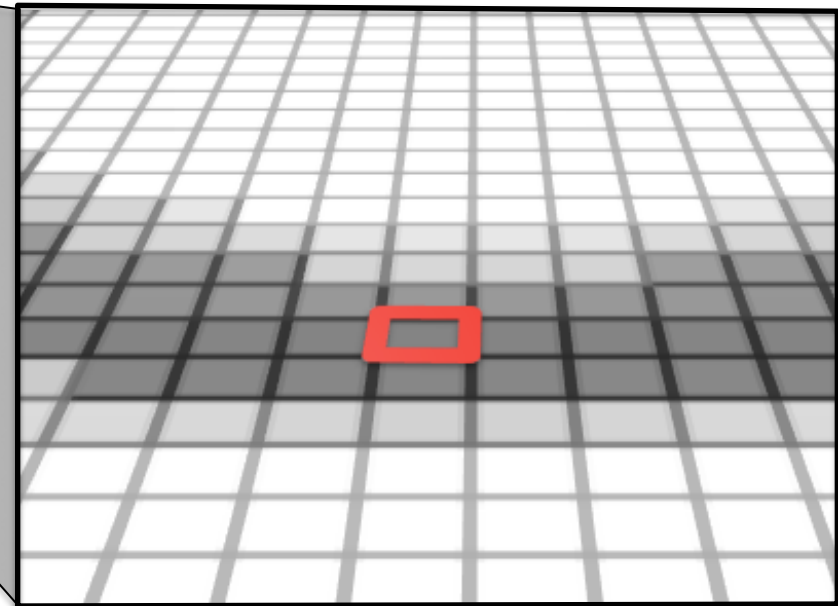
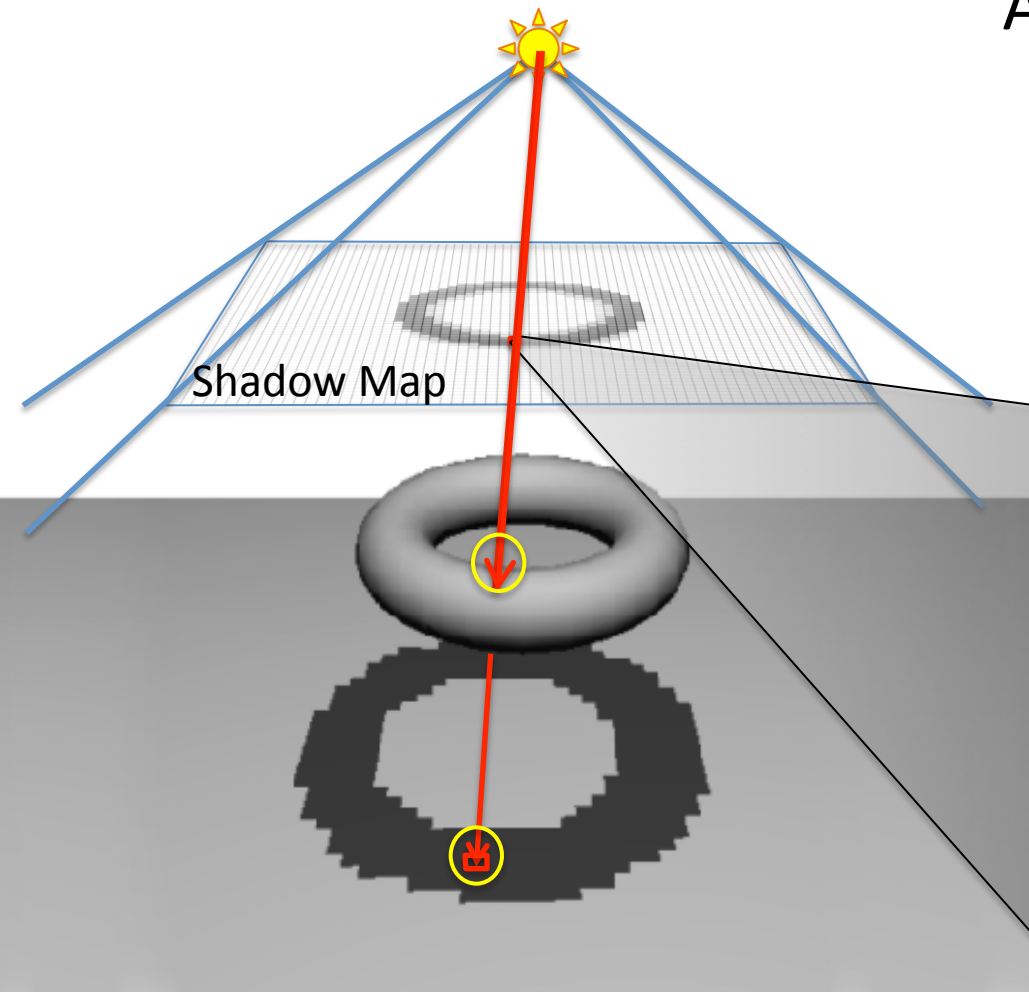


Light's view
(Shadow Map)

Depth Comparison

Render depth image from light

A fragment is in shadow if its depth is greater than the corresponding depth value in the shadow map



Camera's view

Shadow Maps

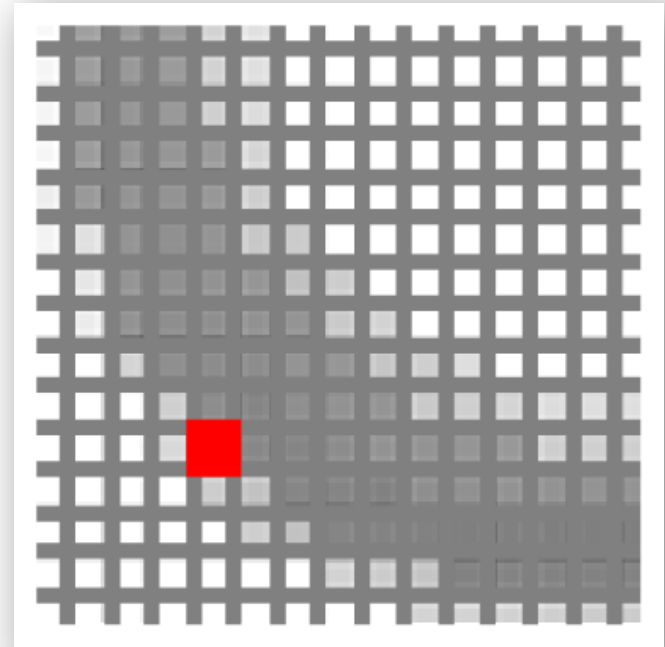
- Pros
 - Very efficient: “This is as fast as it gets”
- Cons...

Shadow Maps - Problems

- Low Shadow Map resolution results in jagged shadows



from viewpoint



from light

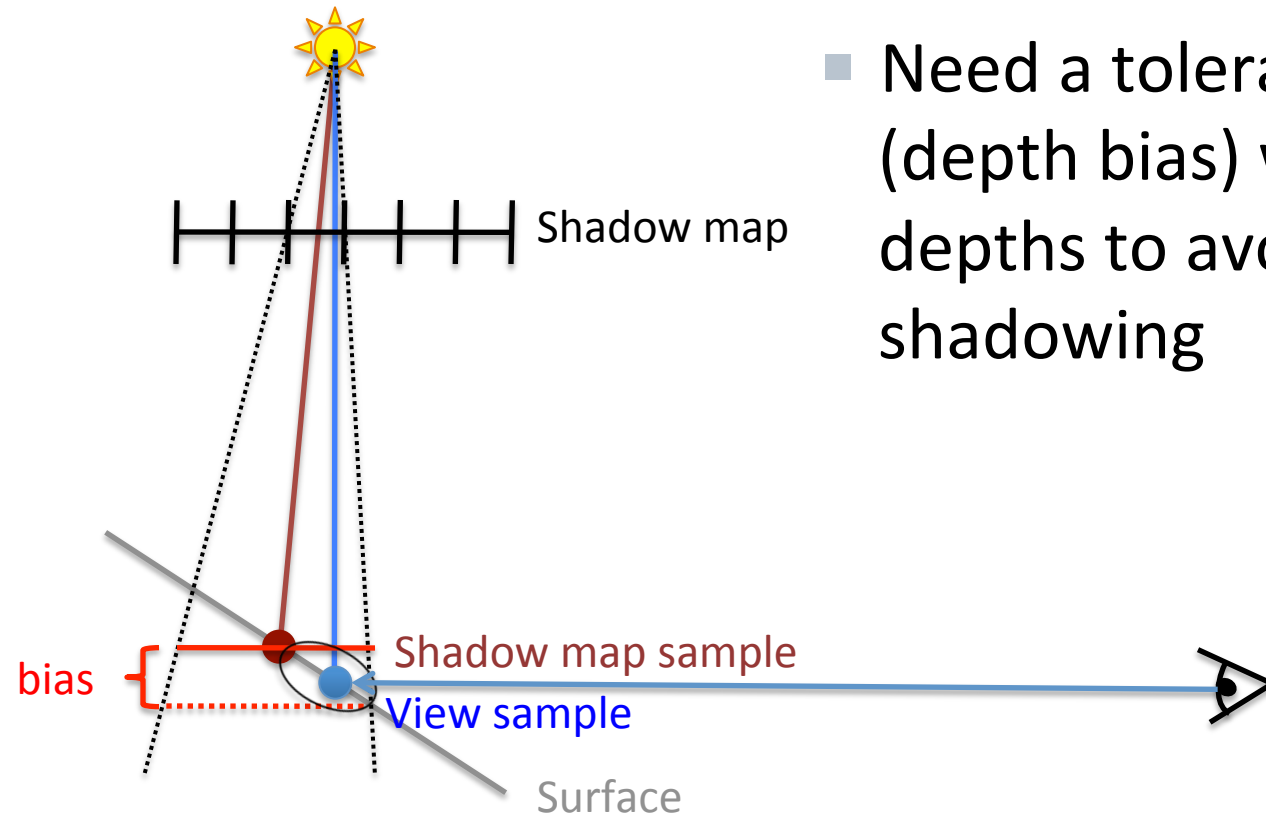
Shadow Maps - Problems

In addition:

- A tolerance threshold (bias) needs to be tuned for each scene for the depth comparison

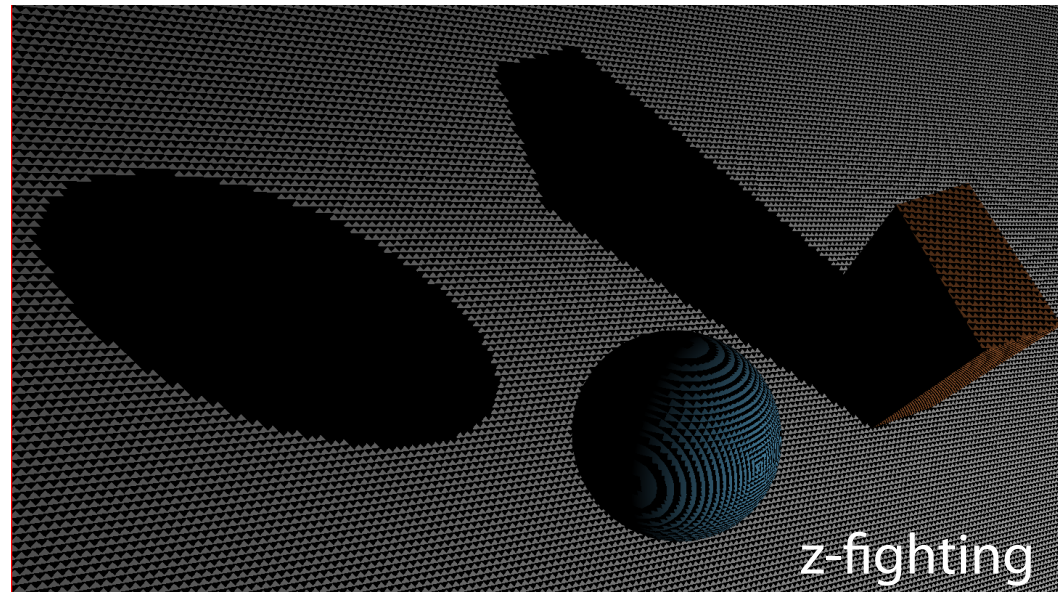
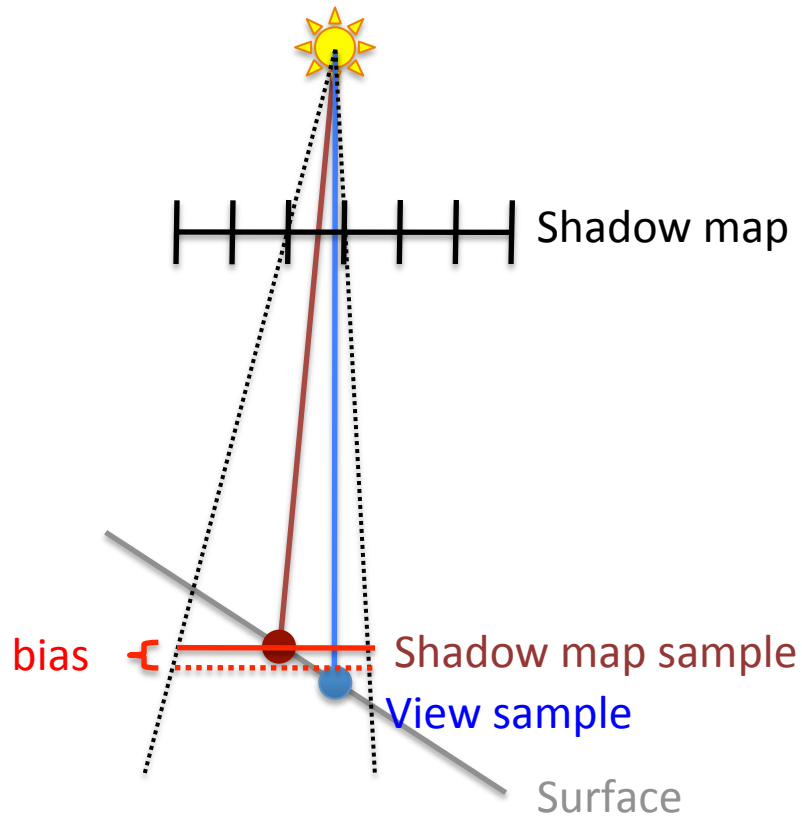
Bias

- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing



Bias

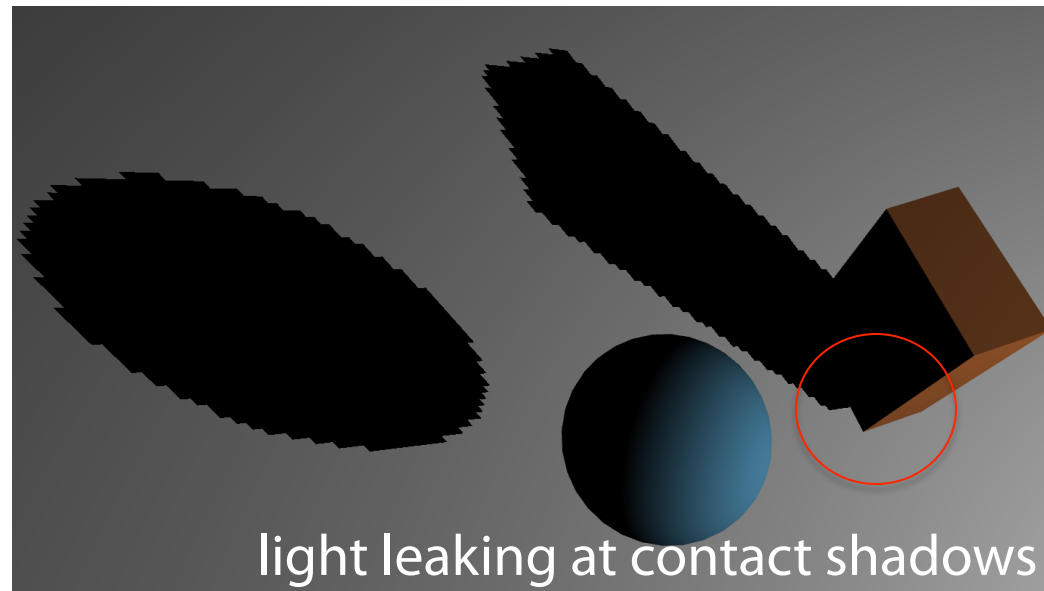
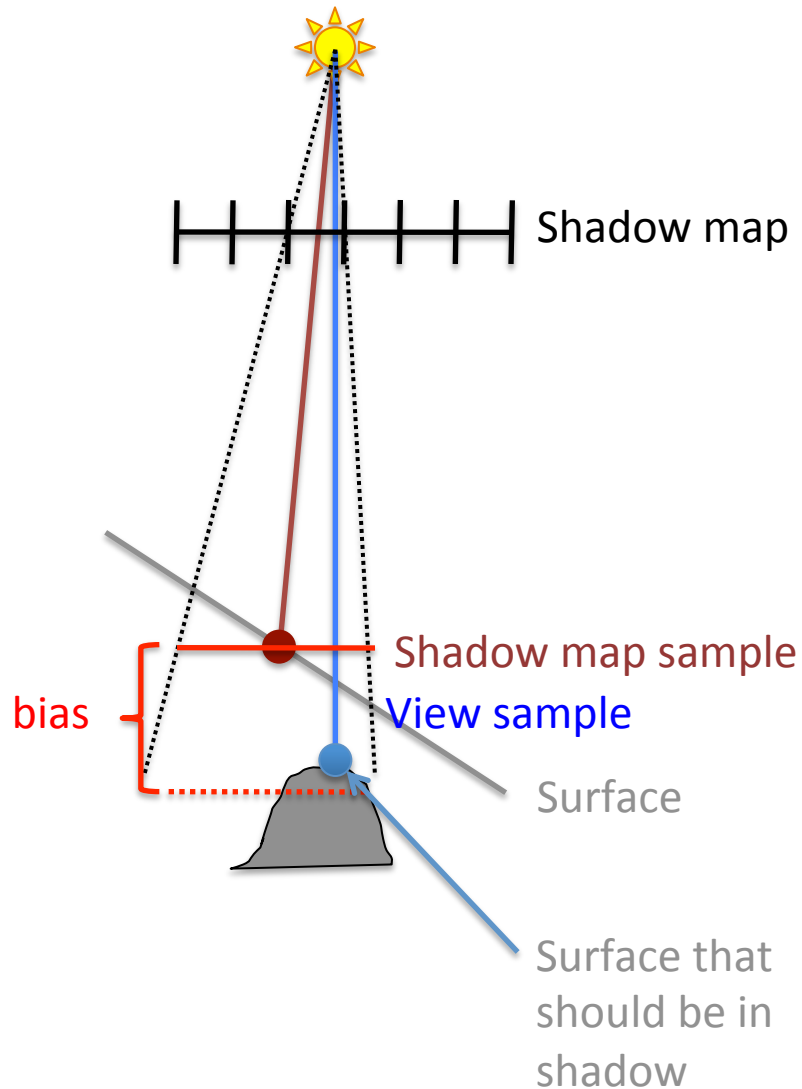
- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing



z-fighting

Bias

- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing



Implementing Shadow Maps

- See tutorial 6 on how to implement shadow maps in practice, as of 2012 😊.
 - Changes every now and then, but algorithm stayed the same since 1978.

Percentage Closer Filtering

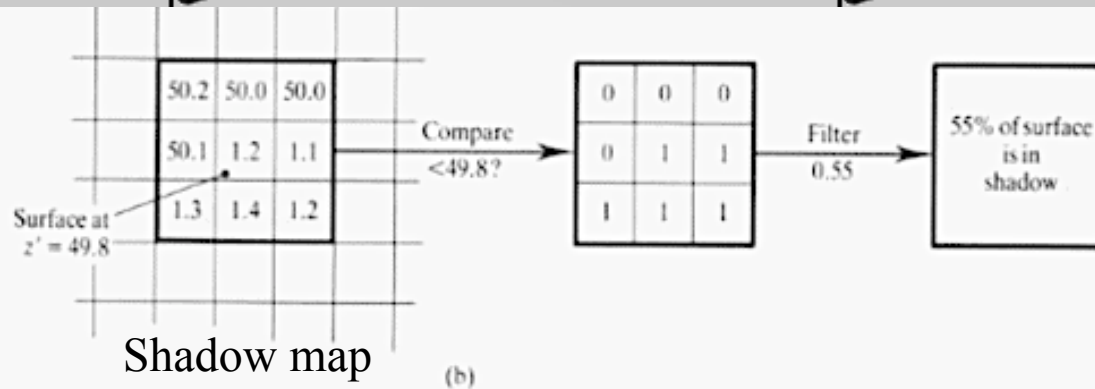
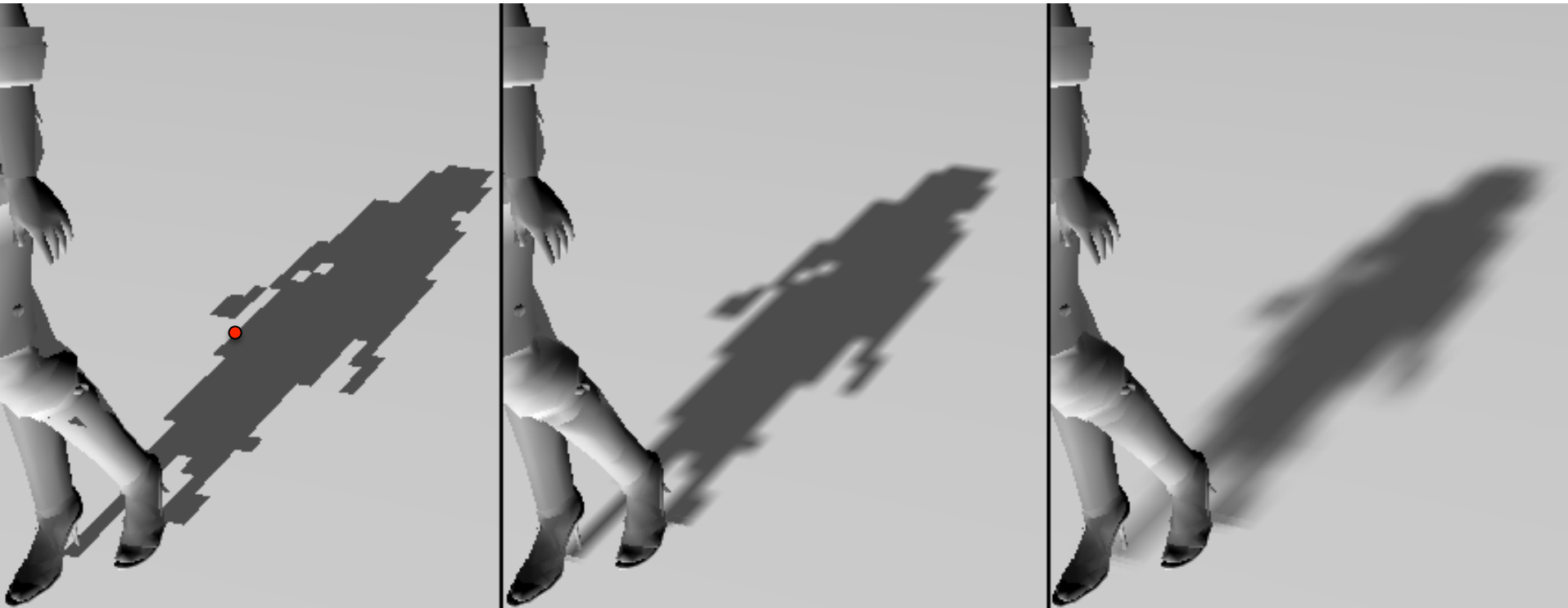


Percentage Closer Filtering

CryTek Soft Shadows

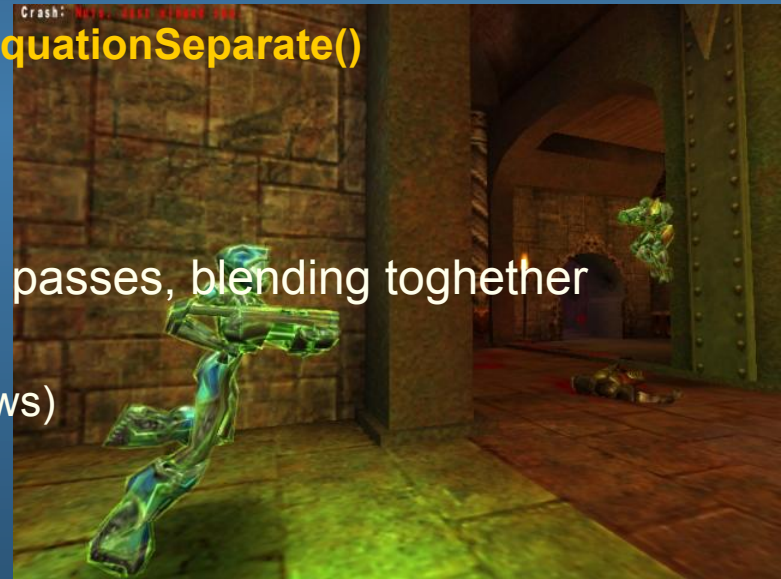


Percentage Closer Filtering



Blending

- Used for
 - Transparency
 - `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`
 - `glBlendEquation()`
 - `glBlendFuncSeparate()` / `glBlendEquationSeparate()`
 - Effects (shadows, reflections)
 - Complex materials
 - Quake3 uses up to 10 rendering passes, blending together contributions such as:
 - Diffuse lighting (for hard shadows)
 - Bump maps
 - Base texture
 - Specular and emissive lighting
 - Volumetric/atmospheric effects
 - Enable with `glEnable(GL_BLEND)`



Example of blending for Motion Blur

Possible with usage of e.g blending to floating point rgb buffer and averaging result before displaying



Image courtesy Brostow and Essa

Misc

Point / Line width

glPointSize(*float size*)

glEnable/Disable(`VERTEX_PROGRAM_POINT_SIZE`)

glLineWidth(*float width*)

glEnable/Disable(`LINE_SMOOTH`)

Polygon rendering

glPolygonMode(*enum face, enum mode*)

– *face*: `FRONT`, `BACK`, `FRONT_AND_BACK`

– *mode*: `POINT`, `LINE`, `FILL`

glPolygonOffset(*float factor, float units*)

glEnable/Disable(*target*)

–`POLYGON_OFFSET_POINT`, `POLYGON_OFFSET_LINE`, `POLYGON_OFFSET_FILL`

Reading Frame Buffers

glReadPixels(*int x, int y, width, height, format, type, void *data*);

glReadBuffer(*enum src*);

–*src*: `NONE`, `FRONT_LEFT`, `FRONT_RIGHT`, `BACK_LEFT`, `BACK_RIGHT`, `FRONT`, `BACK`, `LEFT`, `RIGHT`, `FRONT_AND_BACK`,

–`AUXi` (where *i* is [0, `AUX_BUFFERS` - 1]), `COLOR_ATTACHMENTi` (where *i* is [0, `MAX_COLOR_ATTACHMENTS` - 1])

glBlitFramebuffer(*srcX0, srcY0, srcX1, srcY1, dstX0, dstY0, dstX1, dstY1, bitfield mask, enum filter*);

–*mask*: Bitwise OR of `COLOR_BUFFER_BIT`, `DEPTH_BUFFER_BIT`,

`STENCIL_BUFFER_BIT`

–*filter*: `LINEAR`, `NEAREST`

Buffers

Drawing to Frame Buffers

Selecting a Buffer for Writing :

glDrawBuffer(enum buf)

- *buf*: NONE, FRONT_LEFT, FRONT_RIGHT, BACK_LEFT, BACK_RIGHT, FRONT, BACK, LEFT, RIGHT, FRONT_AND_BACK, COLOR_ATTACHMENT_{*i*} (where *i* is [0, MAX_COLOR_ATTACHMENTS - 1]),
- AUX_{*i*} (where *i* is [0, AUX_BUFFERS - 1])

DrawBuffers(sizei n, const enum *bufs);

- *bufs*: NONE, FRONT_LEFT, FRONT_RIGHT, BACK_LEFT, BACK_RIGHT,
- COLOR_ATTACHMENT_{*i*} (where *i* is [0, MAX_COLOR_ATTACHMENTS - 1]),
- AUX_{*i*} (where *i* is [0, AUX_BUFFERS - 1])

FRAGMENT SHADER

```
void main()
{
    gl_FragData[0] =vec4(1,0,0,1);
    gl_FragData[1] =vec4(1,1,0,1);
}
```

Framebuffer Objects

Binding & Managing Framebuffer Objects (collection of renderbuffers, (<=8 colbuffs))

- **glBindFramebuffer(), glGenFramebuffers(), glDeleteFramebuffers()**

Renderbuffers:

- **BindRenderbuffer(), DeleteRenderBuffers(), glGenRenderBuffers(), glRenderBufferStorage() – w,h,depth/color/stencil**

Attaching renderbuffer to current framebuffer object

- **glFramebufferRenderbuffer()**

Attaching Texture Image to Framebuffer

- **glFramebufferTexture1/2/3D()**

Buffers

- Frame buffer
 - Back/front/left/right – **glDrawBuffers()**
- Depth buffer (z-buffer)
 - For correct depth sorting
 - Instead of BSP-algorithm, painters algorithm...
 - **glDepthFunc(), glDepthMask(false)**
- Stencil buffer
 - Shadow volumes,
 - **glStencilFunc(), glStencilFuncSeparate, glStencilMask, glStencilOp**
- General commands:
 - **glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT)**
 - Specify clearing value: **glClearStencil(), glClearColor(), glClearDepth(default=1)**

Specials



- "Clip planes" (8):

- Fragment shader: `glClipDistance[]`
- `glEnable(GL_CLIP_DISTANCEi)`

- Scissors:

- `glScissor(x,y,w,h)`, `glEnable(GL_SCISSOR_TEST)`

- Finishes all draw calls before CPU-execution continues:

- `glFinish()`

- ~~● Fog: `glFog()`, `glEnable(GL_FOG);`~~



Fragment Operations

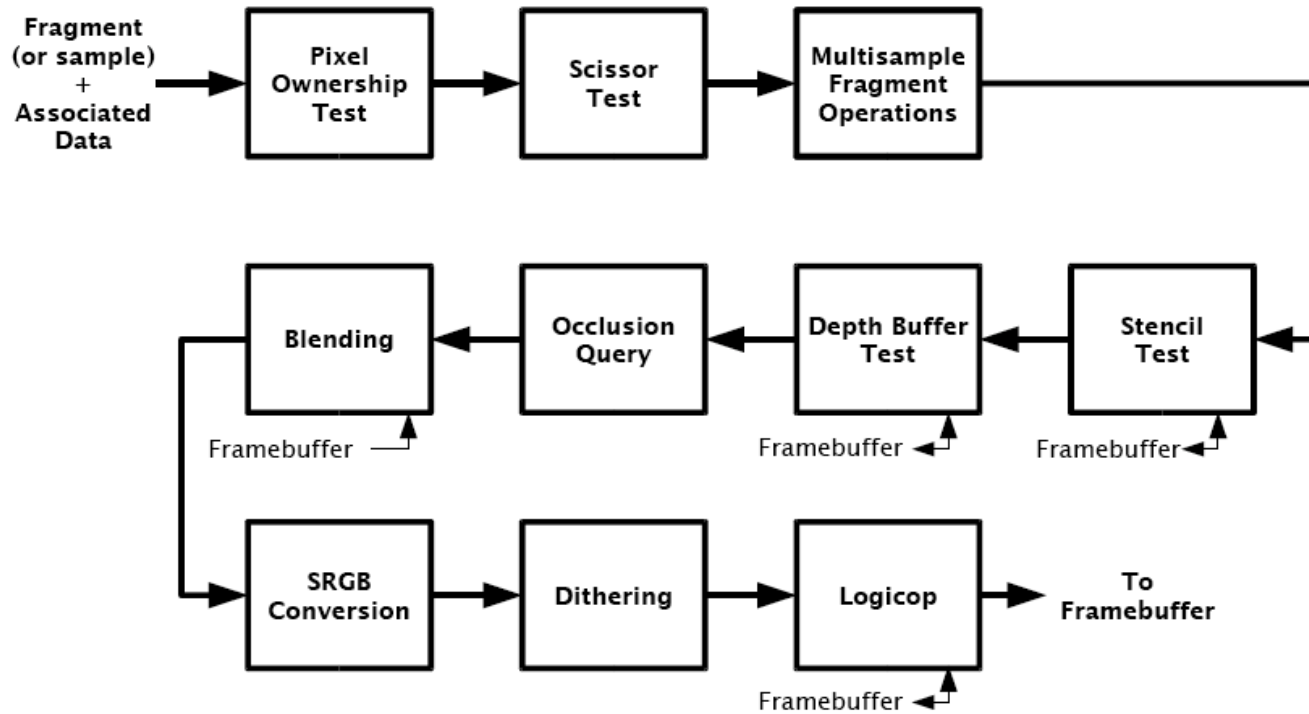


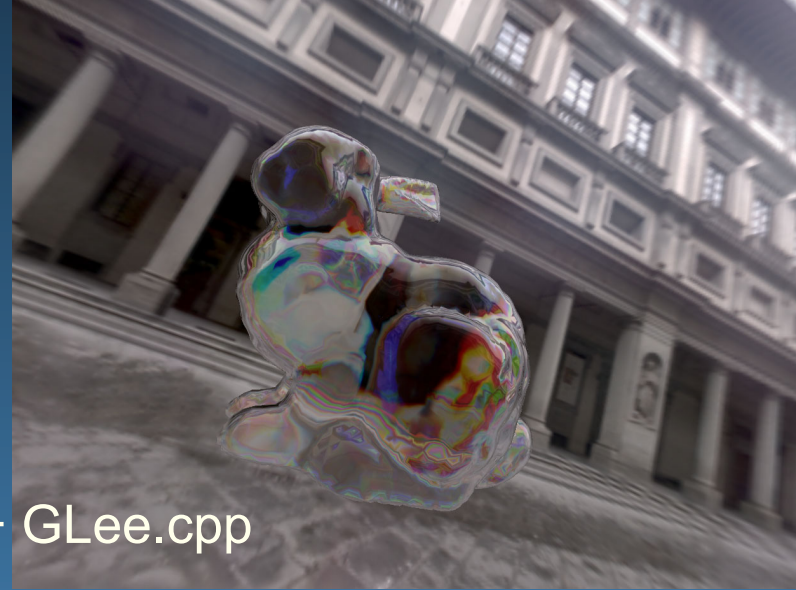
Figure 4.1. Per-fragment operations.

Errors:

- You might find the following code useful:

```
inline CheckGLError()
{
    GLenum errCode;
    const unsigned char* errString;
    if((errCode=glGetError()) != GL_NO_ERROR)
    {
        errString=gluErrorString(errCode);
        printf("OpenGL Error: %s\n", errString);
    }
}
```

Extensions



- `glew.h + glew32.lib/dll` OR `GLee.h + GLee.cpp`
- Or get the extensions manually:
- Check if extension is supported:
`glutExtensionSupported("GL_EXT_framebuffer_sRGB")`
`glutExtensionSupported("GL_EXT_texture_integer")`
- Get address of extension function:
 - `glTexParameterIivEXT = wglGetProcAddress("glTexParameterIivEXT");`
 - `glClearColorIiEXT = wglGetProcAddress("glClearColorIiEXT");`

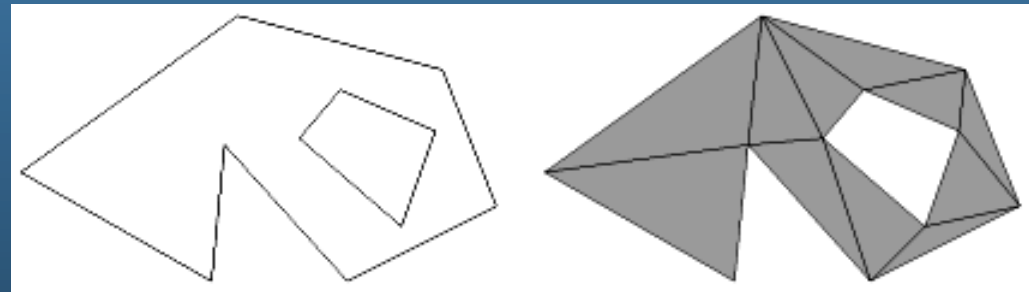
GLU – The OpenGL Graphics System Utility Library

- `#include <GL/glu.h>`. Loads: `glu32.dll` or link with `glu32.lib`
- Support for creating Mip maps
- Matrix manipulation functions (=camera helper functions)
- Polygon Tessellation
 - Creating arbitrary (non-convex) polygons
- Quadrics (2:nd order surfaces)
- NURBS

GLU - Polygon Tesselation

- **The GLU Tesselation Functions**

1. `gluTessBeginPolygon()` begins a new polygon.
2. `gluTessBeginContour()` begins a new contour.
3. `gluTessVertex()` is called repeatedly to pass the vertices to the tessellator.
4. `gluTessEndContour()` ends the contour. If there are more contours in the polygon, continue at Step 2.
5. `gluTessEndPolygon()`



A concave polygon with one hole (left) and the same polygon after tesselation (right)

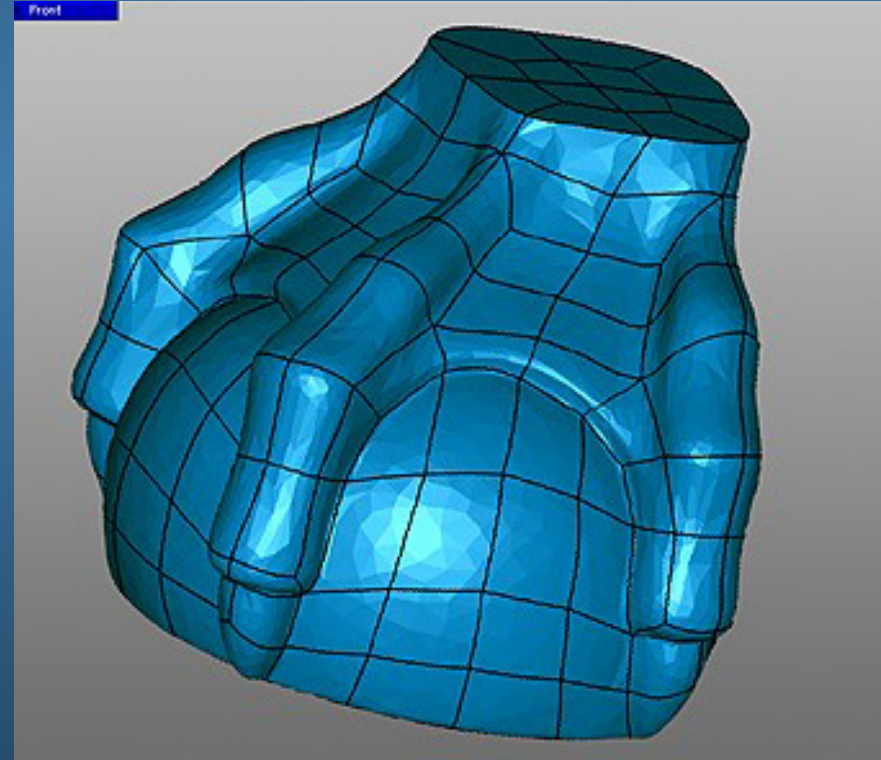
GLU - Quadrics

- To render spheres, cylinders and disks.
 - Example:

```
GLUquadricObj *gQuad;  
gQuad=gluNewQuadric();  
gluQuadricDrawStyle(gQuad, GLU_FILL);  
gluSphere(gQuad,radius, 40,40); // slides, stacks – Draws the sphere
```
 - **gluQuadricNormals()** – **GLU_NONE, GLU_FLAT, GLU_SMOOTH**
 - **gluQuadricTexture()** – **GL_TRUE, GL_FALSE**
 - **gluQuadricOrientation()** – **GLU_OUTSIDE, GLU_INSIDE**
 - **gluQuadricDrawStyle()** – **GLU_FILL, GLU_LINE, GLU_POINT, GLU_SILHOUETTE**
- **gluSphere(), gluDisk(), gluCylinder()**

GLU - NURBS

- See chapter 7 in <http://www.ce.chalmers.se/staff/uffe/glu1.3.pdf> for more information.
- And chapter 24, page 34-38 in "Introduktion till OpenGL" at course homepage



http://www.cse.chalmers.se/edu/course/TDA361/OPENGL_2006.pdf

GLUT – The OpenGL Utility Toolkit

- for creating an OpenGL application with platform independent code.
- `#include <GL/glut.h>`.
 - Links with `glut32.lib` or loads `glut32.dll` (MS Windows).
- Windows, menus, events, text, objects

- <http://www.cse.chalmers.se/~uffe/glut-3.spec.pdf>

GLUT – windows and menus

- Initialization:
 - glutInit(), glutInitDisplayMode(), glutInitWindowPosition(), glutInitWindowSize()
- Start main loop: glutMainLoop()
- Windows:
 - glutCreateWindow, glutCreateSubWindow, glutSetWindow, glutGetWindow, glutDestroyWindow, glutPositionWindow, glutReshapeWindow, glutFullScreen, glutPushWindow, glutPopWindow, glutShowWindow, glutHideWindow, glutIconifyWindow, glutSetWindowTitle, glutSetIconTitle,
 - glutPostRedisplay, glutSwapBuffers, glutSetCursor
- Overlays:
 - glutEstablishOverlay, glutUseLayer, glutRemoveOverlay, glutPostOverlayRedisplay, glutShowOverlay, glutHideOverlay
- Menus:
 - glutCreateMenu, glutSetMenu, glutGetMenu, glutDestroyMenu, glutAddMenuEntry, glutAddSubMenu, glutChangeToMenuEntry, glutChangeToSubMenu, glutRemoveMenuItem, glutAttachMenu, glutDetachMenu

Event Callbacks

– Most common:

- glutDisplayFunc – the scene drawing should be done here
- glutReshapeFunc – on resizing the window. Call **glViewport(0, 0, newWidth, newHeight);**
- glutKeyboardFunc
- glutMouseFunc – mouse buttons
- glutMotionFunc – mouse movements when buttons are pressed
- glutPassiveMotionFunc – when buttons are not pressed
- glutSpecialFunc – for function or direction keys
- glutIdleFunc
- glutTimerFunc

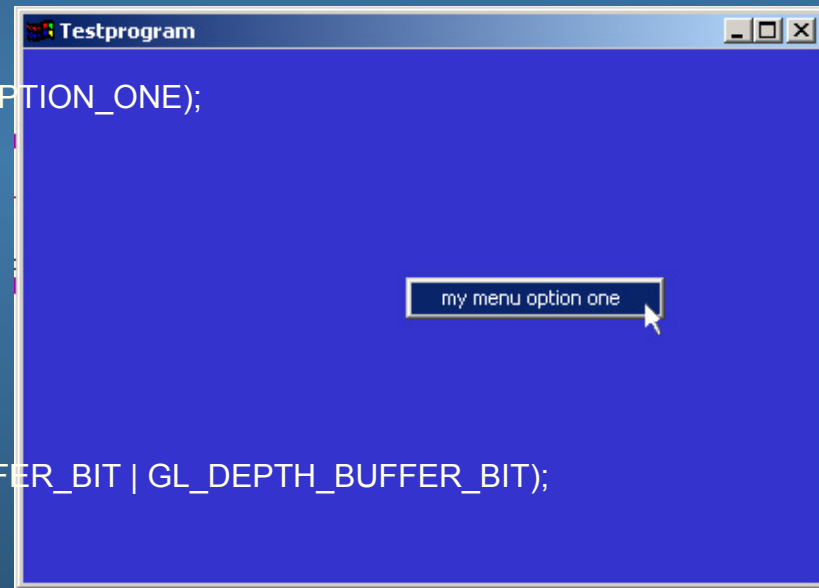
– Not so common:

- glutOverlayDisplayFunc, glutVisibilityFunc, glutEntryFunc, glutSpaceballMotionFunc, glutSpaceballRotateFunc, glutSpaceballButtonFunc, glutButtonBoxFunc, glutDialsFunc, glutTabletMotionFunc, glutTabletButtonFunc, glutMenuStatusFunc,

Program Example

```
#ifdef WIN32 #include <windows.h> #endif
#include <GL/glut.h>
enum {MY_MENU_OPTION_ONE};
int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800,600); glutCreateWindow("Testprogram");
    glutKeyboardFunc(handleKeys); glutSpecialFunc(handleSpecialKeys); glutDisplayFunc(display);
    glutMouseFunc(mouse); glutMotionFunc(motion); glutReshapeFunc(reshape); glutIdleFunc( idle );

    glutCreateMenu(menu);
    glutAddMenuEntry("my menu option one", MY_MENU_OPTION_ONE);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutMainLoop();
}
void idle() {
    ... do animation computations ...
    glutPostRedisplay();
}
void display() {
    glClearColor(0.2,0.2,0.8,1.0); glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    ... draw the scene ...
    glutSwapBuffers(); // swap front and back buffer
}
void menu(int value) {
    switch(value) {
    case MY_MENU_OPTION_ONE:
        ... do some stuff ...
    }
}
}
```



Obsolete for OpenGL 3
but you can still use it

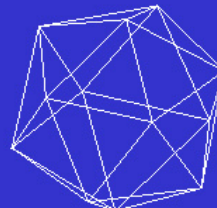
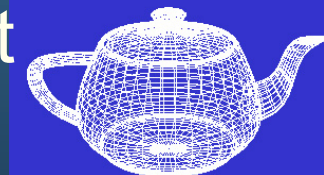
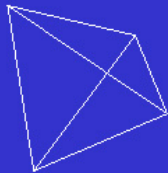
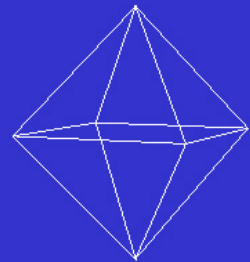
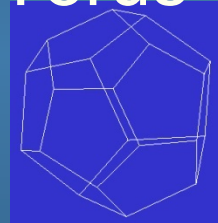
Text

- Commands:
 - glutBitmapCharacter, glutStrokeCharacter,
- Example:

```
void print(char* str) {
    glMatrixMode(GL_PROJECTION); glPushMatrix();
    gluOrtho2D(0, mWinWidth, 0, mWinHeight);
    glMatrixMode(GL_MODELVIEW); glPushMatrix();
    glLoadIdentity();
    glColor3f(1,0,0); // set red text
    glRasterPos2f(10, 10); // origin is lower left window corner
    int len=strlen(str);
    for(int i=0; i<len; i++)
        glutBitmapCharacter(GLUT_BITMAP_8_BY_13, str[i]);
    glMatrixMode(GL_MODELVIEW); glPopMatrix();
    glMatrixMode(GL_PROJECTION); glPopMatrix();
}
```


Predefined Objects

- glutSolidSphere, glutWireSphere
- glutSolidCone, glutWireCone
- glutSolidCube, glutWireCube
- glutSolidTorus, glutWireTorus
- glutSolidDodecahedron, glutWireDodecahedron
- glutSolidOctahedron, glutWireOctahedron
- glutSolidTetrahedron, glutWireTetrahedron
- glutSolidIcosahedron, glutWireIcosahedron
- glutSolidTeapot, glutWireTeapot



Exam Questions

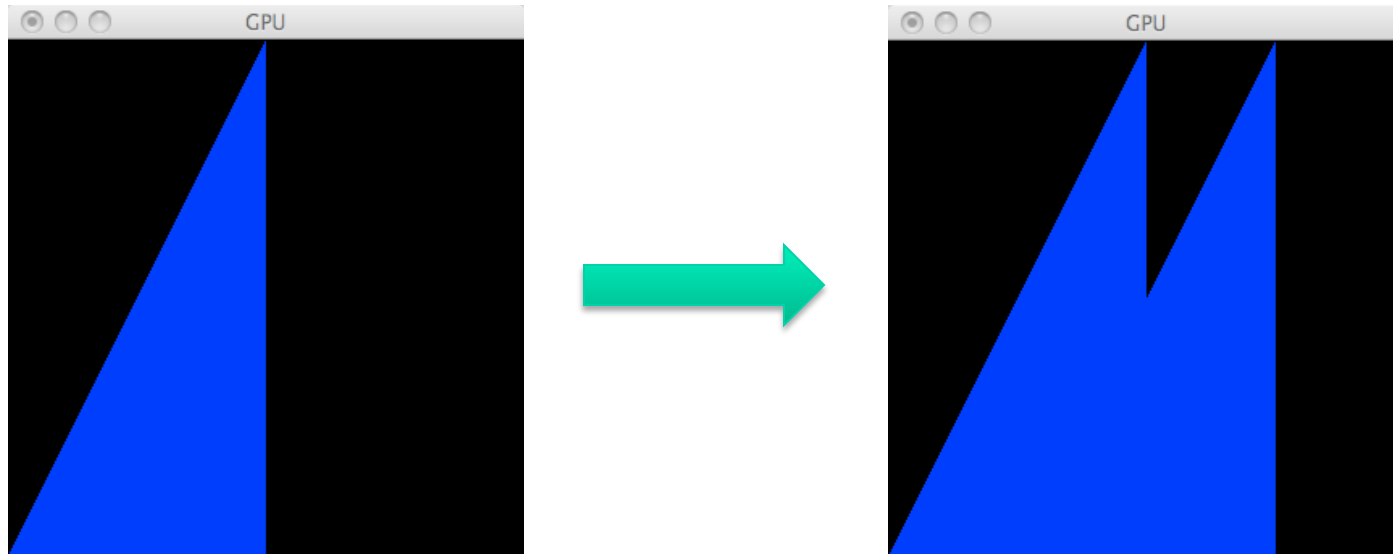
- principles of a real-time rendering API like OpenGL
 - E.g. high level functionality
 - Shadow Maps
 - Types of buffers
 - How do you achieve transparency?
 - (Adding clip planes to the standard unit cube)
 - What defines what is the back and front side of a triangle?

Hunter ate Ranger's rocket
Wrote screenshots/shot0147.tga

**END OF
OPENGL,
GLU AND
GLUT
LECTURE**



Simple Geometry shader demo



Geometry shader

```
#version 120
#extension GL_EXT_geometry_shader4 : enable
void main(void){
    //Pass-thru vertices!
    for(i=0; i< gl_VerticesIn; i++){
        gl_Position = gl_PositionIn[i];
        EmitVertex();
    }
    EndPrimitive();

    //New piece of geometry! Add translation
    for(i=0; i< gl_VerticesIn; i++){
        gl_Position = gl_PositionIn[i];
        gl_Position.xy += vec2(0.5,0);
        EmitVertex();
    }
    EndPrimitive();
}
```

Loading the shaders

```
void setShaders() {
    GLuint v = glCreateShader(GL_VERTEX_SHADER);
    GLuint f = glCreateShader(GL_FRAGMENT_SHADER);
    GLuint g = glCreateShader(GL_GEOMETRY_SHADER_EXT);

    char * vs = textFileRead("toon.vert");
    char * fs = textFileRead("toon.frag");
    char * gs = textFileRead("toon.geom");

    glShaderSource(v, 1, (const char **) &vs, NULL);
    glShaderSource(f, 1, (const char **) &fs, NULL);
    glShaderSource(g, 1, (const char **) &gs, NULL);
    free(vs);free(fs);free(gs);

    glCompileShader(v); glCompileShader(f); glCompileShader(g);
    GLuint p = glCreateProgram();
    glAttachShader(p,f); glAttachShader(p,v); glAttachShader(p,g);

    glProgramParameteriEXT(p, GL_GEOMETRY_INPUT_TYPE_EXT, GL_TRIANGLES);
    glProgramParameteriEXT(p, GL_GEOMETRY_OUTPUT_TYPE_EXT, GL_TRIANGLES);
    GLint temp;
    glGetIntegerv(GL_MAX_GEOMETRY_OUTPUT_VERTICES_EXT, &temp);
    glProgramParameteriEXT(p, GL_GEOMETRY_VERTICES_OUT_EXT, temp);

    glLinkProgram(p);
    glUseProgram(p); // 0 disables vertex/fragment shaders
}
```