OpenGL
- a quick guide

Ulf Assarsson
Department of Computer Engineering
Chalmers University of Technology

Reading Material

MUST read
- These slides
- OH 257-263 OpenGL Extensions
- OH 292-297 Stencil buffer, polygon mode, tesselation
- OH 328-332 gluUnproject

- OH 51-64 Scenegraphs

OpenGL vs Direct3D

- Direct3D
  - Microsoft
  - Common for games
  - "Adapted to graphics hardware evolution"
  - (Now after many upgrades very similar to OpenGL)

Direct3D messy to program version 3.0 – 6.0.

- OpenGL
  - SGI
  - "Precede the hardware evolution"
  - Operation system independent
  - Window system independent
  - Industry, games (Quake – thanks John Carmack)
  - 1992
  - Extendable, stable, better design,

Probably why OpenGL still exists

Overview of today's OpenGL lecture

- OpenGL
  - Specifying vertices and polygons
  - Coordinate transformation, viewport, camera
  - Lighting and colors
  - Texturing
  - Blending
  - Buffers (frame b/f/l/r, depth, alpha-channel, stencil, acc)
  - Misc: hint, flush, finish, fog
- GLU – The OpenGL Graphics System Utility Library
- GLUT – The OpenGL Utility Toolkit
  - Windows and menus
  - Callbacks for events
  - Text support
  - Predefined Objects

Example of Motion Blur

Possible with usage of e.g. the accumulation buffer

Image courtesy Brostow and Essa

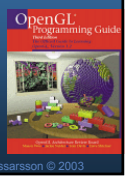
OpenGL – links

- Home page: www.opengl.org
- Sample code: http://www.opengl.org/resources/

- OpenGL specification: http://www.ce.chalmers.se/~uffe/glspec13.pdf
- GLU specification: http://www.ce.chalmers.se/~uffe/glu1.3.pdf
- GLUT specification: http://www.ce.chalmers.se/~uffe/glut-3.spec.pdf

ALSO ON COURSE HOME PAGE:
http://www.ce.chalmers.se/edu/course/TDA360/

- Programmers Manual and Reference Manual: http://www.opengl.org/documentation/red_book/



Include

- #include <GL/gl.h>
- Links with OpenGL32.lib (MS Windows)
- glew.h / glew32.lib / glew32.dll
- GLee.h / GLee.cpp
- http://www.lighthouse3d.com/opengl/

Specifying vertices and polygons

- OpenGL is a state machine. Commands typically change the current state
- Multiple calling formats for the commands: `void glVertex{234}{sifd}( 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.

- Display lists are created with surrounding glNewList() and glEndList(). (Fast)

```
int dlist=glGenLists(1);                    // generate one display list number
glNewList(dlist,GL_COMPILE);                // indicates start of display list
    glutSolidSphere(radius, 20, 20);        // or any object specification
glEndList();                                // indicates end of display list
```

Display lists are then drawn with: glCallList(dlist);

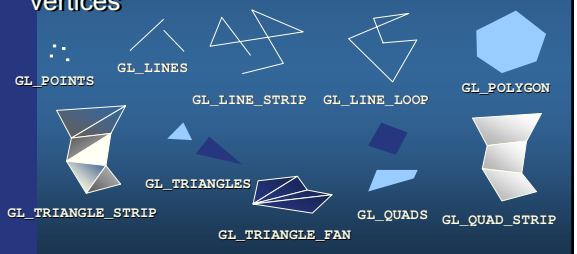
Idea: Objects are stored internally in the most efficient format

- Vertex Arrays (Fast):

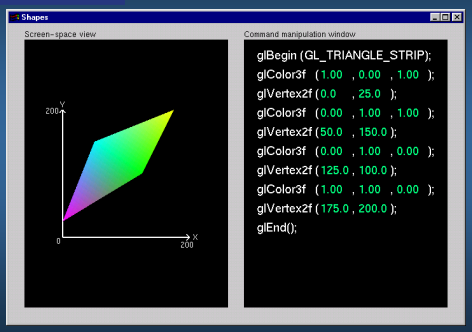
```
void glTexCoordPointer( int size, enum type, sizei stride, void *pointer );   //Defines an array of texture coordinates
void glColorPointer( int size, enum type, sizei stride, void *pointer );      //Defines an array of colors
(void glIndexPointer( enum type, sizei stride, void *pointer );               //Defines an array of color indices)
void glNormalPointer( enum type, sizei stride,void *pointer );                //Defines an array of normals
void glVertexPointer( int size, enum type, sizei stride,void *pointer );      //Defines an array of vertices
void glEnableClientState( enum array ); void glDisableClientState( enum array ); with array set to TEXTURE_COORD_ARRAY,
    COLOR_ARRAY, INDEX_ARRAY, NORMAL_ARRAY, or VERTEX_ARRAY,
void glArrayElement(int i );          // Transfers the ith element of every enabled array to OpenGL
void glDrawArrays (mode, first, count) ;   // Renders multiple primitives from the enabled arrays
void glDrawElements(mode, count, type, *indices );   // Same as glDrawArrays, but takes array elements pointed to by indices.
void glInterleavedArrays (format; stride; pointer) ;   // One array with vertices, normals, textures and/colors interleaved
```


OpenGL Geometric Primitives

- All geometric primitives are specified by vertices



Example:



Example of how to use glDrawArrays

```
glTexCoordPointer(2,GL_FLOAT, 0, *texturepointer );   //Specifies an array of texture
                                                      // coordinates
glColorPointer(4, GL_FLOAT, 0, *colorpointer );       //Specifies an array of colors
glNormalPointer(GL_FLOAT, 0, *normalpointer );        //Specifies an array of normals
glVertexPointer(3, GL_FLOAT, 0,void *vertexpointer ); //Specifies an array of vertices
glPushClientAttrib(GL_CLIENT_ALL_ATTRIB_BITS);        //Save current client state
glEnableClientState(TEXTURE_COORD_ARRAY);             //Enable all arrays
glEnableClientState(COLOR_ARRAY);
glEnableClientState(NORMAL_ARRAY);
glEnableClientState(VERTEX_ARRAY);
glDrawArrays (GL_TRIANGLES; 0; numTriangles) ;        //Renders all triangles
glPopClientAttrib();                                  //Restore the client state
```


Vertex order

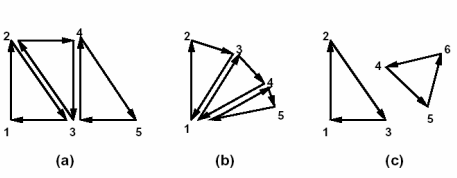


Figure 2.4. (a) A triangle strip. (b) A triangle fan. (c) Independent triangles. The numbers give the sequencing of the vertices between **Begin** and **End**. Note that in (a) and (b) triangle edge ordering is determined by the first triangle, while in (c) the order of each triangle's edges is independent of the other triangles.

Note: Vertex order indicates that all these triangles are backfacing with CCW-ordering (default for OpenGL) for front facing triangles, except for the 2:nd triangle in (c).

Quad formats

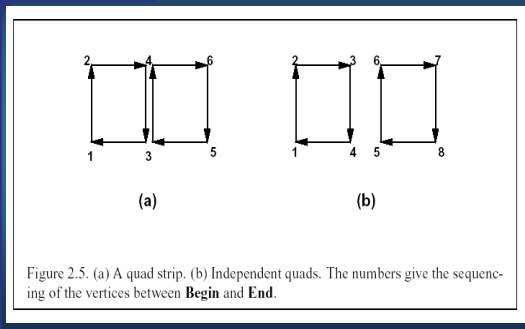


Figure 2.5. (a) A quad strip. (b) Independent quads. The numbers give the sequencing of the vertices between **Begin** and **End**.

Coordinate transformations

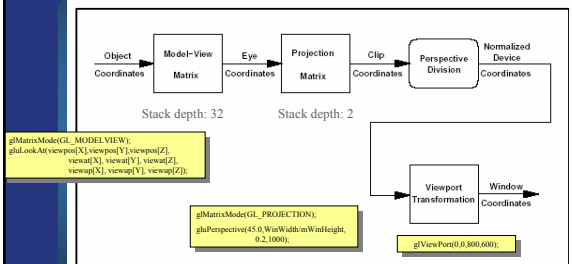
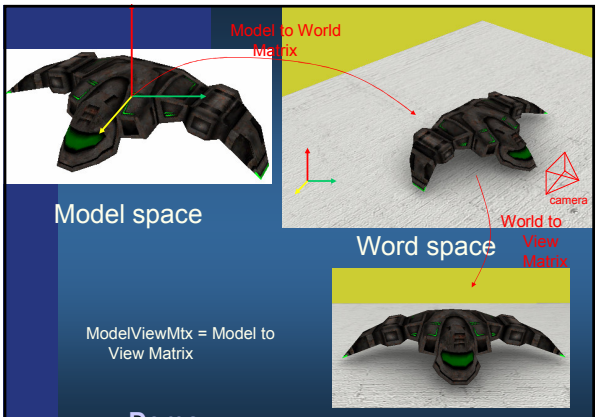


Figure 2.6. Vertex transformation sequence.



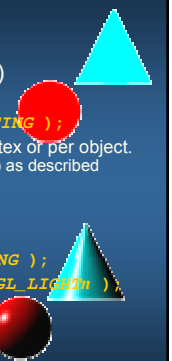
ModelViewMtx = Model to View Matrix

Demo

Matrix and Stack Operations

- Matrix Operations:
  - `glMatrixMode( GL_MODELVIEW or GL_PROJECTION )`
  - `glLoadIdentity(), glMultMatrix()`
  - `glRotate(),glTranslate(), glScale(), (glFrustum(), glOrtho())`
  - **GLU Helper functions:** `gluPerspective(), gluLookAt, gluOrtho2D()` (good for 2D rendering like text)
- Stack Operations:
  - `glPushMatrix(), glPopMatrix()`
  - `glPushAttrib(), glPopAttrib()`

Lighting and Colors



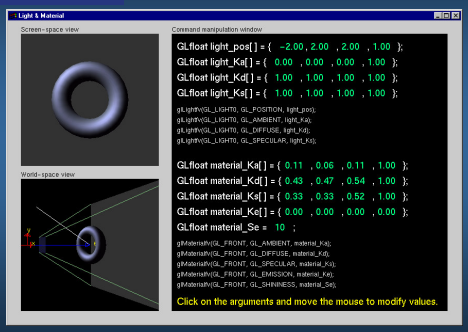
- glColor4f(r,g,b,a), glColor3f(r,g,b)
  - Used when lighting is disabled:
  - Disable with `glDisable( GL_LIGHTING );`
  - Could be changed for instance per vertex or per object. Can also be specified with glColorPointer() as described previously
- glMaterialfv()
  - Used when lighting is enabled.
  - Enable with `glEnable( GL_LIGHTING );`
  - Must also enable lights: `glEnable( GL_LIGHT0 );`
  - Example:
    - glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, float rgba[4])
    - glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, float rgba[4])
    - glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, float rgba[4])
    - glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, float rgba[4])
    - glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 30)

glMaterialfv()

- material components

`GL_DIFFUSE`	Base color
`GL_SPECULAR`	Highlight Color
`GL_AMBIENT`	Low-light Color
`GL_EMISSION`	Glow Color
`GL_SHININESS`	Surface Smoothness

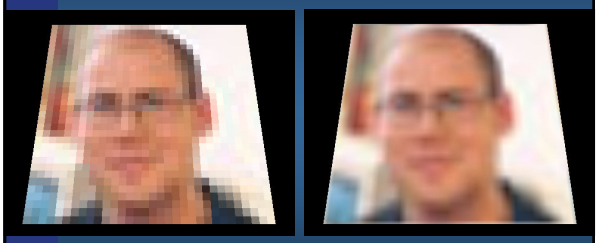
Example of using lighting



Texture Mapping

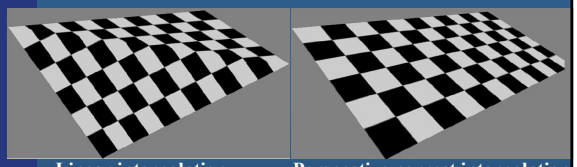
- Three steps
  1. specify texture
     - read or generate image
     - assign to texture – `glGenTextures()`, `glBindTexture()`, `gluBuild2DMipMaps()`
  2. assign texture coordinates to vertices
  3. specify texture parameters
     - set texture filter – `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,… )`
     - set texture function – `glTexEnvf(GL_TEXTURE_ENV, GL_MODULATE / GL_DECAL / GL_BLEND / GL_ADD or GL_COMBINE )`
     - set texture wrap mode – `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,…)`
     - set optional perspective correction hint – `glHint( GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST)`
     - bind texture object – `glBindTexture()`
     - enable texturing – `glEnable(GL_TEXTURE_2D)`
     - supply texture coordinates for vertex – `glTexCoord2f()`, `glTexCoord3f()`, `glTexCoord4f()`
       - coordinates can also be generated: `glTexGen(GL_OBJECT_LINEAR/GL_EYE_LINEAR/GL_SPHERE_MAP)`
       - `glEnable(GL_TEX_GEN_S/T/R/Q)`

Examples of filtering



Nearest Linear

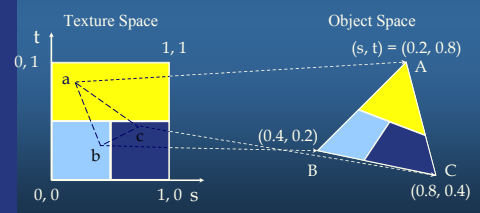
Perspective correct texturing



Linear interpolation Perspective-correct interpolation

Assigning Texture coordinates - glTexCoord()

- Based on parametric texture coordinates
- `glTexCoord2f()` specified at each vertex



Specifying a Texture: Other Methods

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

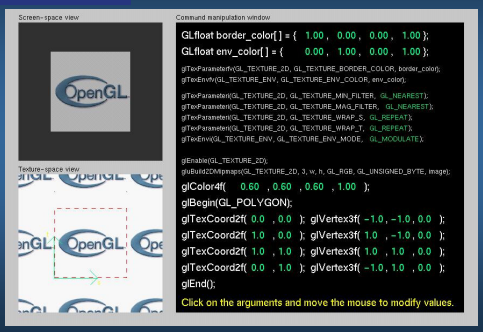
```
glCopyTexImage1D(...)
glCopyTexImage2D(...)
```

- Modify part of a defined texture

```
glTexSubImage1D(...)
glTexSubImage2D(...)
glTexSubImage3D(...)
```

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

Example of using texturing



Multitextures

- glActiveTexture (GL_TEXTURE0);

```
glBindTexture(GL_TEXTURE_2D, m_woodID);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE, GL_MODULATE);

glEnable(GL_TEXTURE_2D);
```

glActiveTexture (GL_TEXTURE1);
....

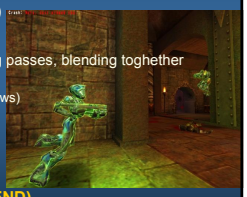
- void **MultiTexCoord**f1234gfsifdg(enum *texture*,T *coords*)
  - glMultiTexCoord2f (GL_TEXTURE0, x, y);
  - glMultiTexCoord2f (GL_TEXTURE1, x, y);

Reflections with environment mapping

- Uses the active texture as an environment map
- Enable with:
  - `glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);`
  - `glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);`
  - `glEnable(GL_TEXTURE_GEN_S);`
  - `glEnable(GL_TEXTURE_GEN_T);`
- Cube mapping in OpenGL1.3
  - See glSpec13.pdf (link on homepage) or glSpec14.pdf on the web

Blending

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



Buffers

- Frame buffer
  - Back/front/left/right – **glDrawBuffers()**
  - Alpha channel: **glAlphaFunc()**
- Depth buffer (z-buffer)
  - For correct depth sorting
  - Instead of BSP-algorithm, painters algorithm…
  - **glDepthFunc(), glDepthMask, glDepthRange()**
- Stencil buffer
  - Shadow volumes,
  - **glStencilFunc(), glStencilMask, glStencilOp() –see www.msdn.com**
- Accumulation buffer
  - **glAccum(),** GL_LOAD, GL_ACCUM, GL_ADD, GL_MULT, GL_RETURN
- General commands:
  - **glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_ACCUM_BUFFER_BIT | GL_STENCIL_BUFFER_BIT)**
  - Specify clearing value: **glClearAccum(), glClearStencil(), glClearColor()**

Specials

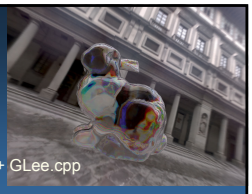
- Clip planes (6):
  - `glClipPlane()`
  - `glEnable(GL_CLIP_PLANEi)`
- Fog: `glFog(), glEnable(GL_FOG);`
- Reading the contents of the buffers:
  - `glReadBuffer(), glReadPixels(x,y,…)`
- Scissors:
  - `glScissor(x,y,w,h), glEnable(GL_SCISSOR_TEST)`
- `glHint(), glFlush, glFinish()`

Errors:

- Feedback for error detection:
  - `glPassThrough()`, `glRenderMode(GL_FEEDBACK)`, `glFeedbackBuffer()`.
- 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
- Check if extension is supported:
  `glutExtensionSupported("GL_ARB_multitexture")`
- Get address of extension function:
  - `glMultiTexCoord2iARB = wglGetProcAddress("glMultiTexCoord2iARB");`
  - `glMultiTexCoord3fARB = wglGetProcAddress("glMultiTexCoord3fARB");`
  - `glMultiTexCoord3fvARB = wglGetProcAddress("glMultiTexCoord3fvARB");`
  - `glActiveTextureARB = wglGetProcAddress("glActiveTextureARB");`
  - `glClientActiveTextureARB = wglGetProcAddress("glClientActiveTextureARB");`
- Could be a bit messy with types. Hint: have a peek at an nVidia demo or use it as a templates.

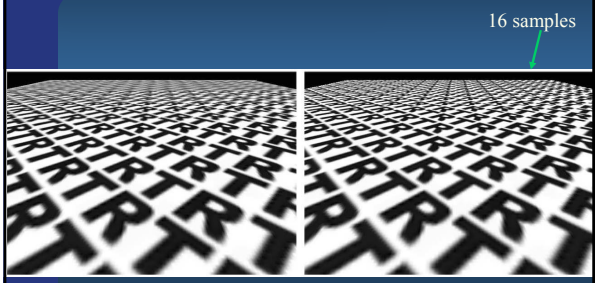
GLU – The OpenGL Graphics System Utility Library

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

GLU - Mip Mapping Support

- Manual image scaling: gluScaleImage()
- Completely automatic (recommended):
  - **gluBuild1DMipmaps(), gluBuild2DMipmaps(), gluBuild3DMipmaps()**
    - Calls **glTexImage[1/2/3]D()** for every n:th needed scale ($2^n * 2^n$)
  - To only affect some mipmap levels, use: **gluBuild[1/2/3]DMipmapLevels()**
- Or
  - glTexParameteri(GL_TEXTURE_2D, GL_GENERATE_MIPMAPS_SGIS)
  - glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, 16)

Example of anisotropic filtering



GLU - Matrix manipulation (Camera)

- **gluOrtho2D(…)** – creates an orthographic camera. Very useful for text or 2D rendering. Used with **glMatrixMode(GL_PROJECTION)**.
- **gluPerspective(…)** – creates a perspective camera with given field of view, aspect ratio, near and far plane. Used with **glMatrixMode(GL_PROJECTION)**.
- **gluLookAt(…)** – creates a viewing matrix for a specified eye position, lookat position and up vector. Used with **glMatrixMode(GL_MODELVIEW)**.

GLU - Matrix manipulation (continued)

- **gluPickMatrix(…) –** help for picking.
  - Use together with glRenderMode(GL_SELECT) and glSelectBuffer() - uses glInitName(), glPushName(), glPopName(), glLoadName() .
  - Perhaps not the best way to do picking. See http://www.ce.chalmers.se/staff/uffe/glu1.3.pdf and http://www.cs.pitt.edu/~panos/teaching/d1566/files/picking.pdf for more information.
  - If many objects are used, raytracing in a spatial hierarchy is probably faster.
- **gluProject(…) –** projects from object space to screen space.
- **gluUnproject(…) –** projects from screen space to object space. Good for creating a pick ray. Uses near clipping plane to define a 3D point from screen space.

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 tesselator.**
  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

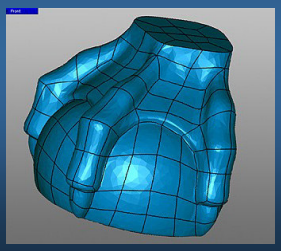
- To render spheres, cylinders and disks.
  - Example:

```
GLUquadricObj *gQuad;
gQuad=gluNewQuadric();
gluQuadricDrawStyle(gQuad, GLU_FILL);
gluSphere(gQuad,radius, 40,40); // slides, stacks
```

  - **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.
- Och kap 24, s34-38 i ”Introduktion till OpenGL” på kurshem-sidan



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.ce.chalmers.se/~uffe/glut-3.spec.pdf

GLUT – windows and menus

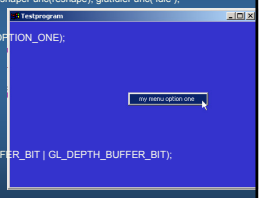
- Initialization:
  - glutInit(), glutInitDisplayMode(), glutInitWindowPosition(), glutInitWindowSize()
- Start main loop: glutMainLoop()
- Windows:
  - glutCreateWindow, glutCreateSubWindow, glutSetWindow, glutGetWindow, glutDestroyWindow, glutPositionWindow, glutReshapeWindow, glutFullScreen, glutPushWindow, 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(menus);
    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 menus(int value) {
    switch(value) {
    case MY_MENU_OPTION_ONE:
            … do some stuff …
    }
}
```



Text

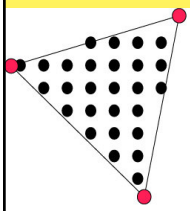
- Commands:
  - glutBitmapCharacter, glutBitmapWidth, glutStrokeCharacter, glutStrokeWidth
- Example:

```
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

What is vertex and fragment (pixel) shaders?



- Memory: Texture memory (read + write) typically 64-256 Mb
- Program size: 64, 1024 or unlimited instructions
- Instructions: mul, rcp, mov,dp, rsq, exp, log, cmp, jnz…
- 32 bits processors, usually SIMD

- For each vertex, a vertex program (vertex shader) is executed
- For each fragment (pixel) a fragment program (fragment shader) is executed

Cg - "C for Graphics" (NVIDIA)

```
if (slice >= 0.0h) {
    half gradedEta = BallData.ETA;
    gradedEta = 1.0h/gradedEta; // test hack
    half3 faceColor = BgColor; // blown out - go to BG color
    half c1 = dot(-Vn,Nf);
    half cs2 = 1.0h-gradedEta*gradedEta*(1.0h-c1*c1);

    if (cs2 >= 0.0h) {
        half3 refVector = gradedEta*Vn+((gradedEta*c1-sqrt(cs2))*Nf);
        // now let's intersect with the iris plane
        half irisT = intersect_plane(IN.OPosition,refVector,planeEquation);
        half fadeT = irisT * BallData.LENS_DENSITY;
        fadeT = fadeT * fadeT;
        faceColor = DiffPupil.xxx;  // temporary (?)
        if (irisT > 0) {
            half3 irisPoint = IN.OPosition + irisT*refVector;
            half3 irisST = (irisScale*irisPoint) + half3(0.0h,0.5h,0.5h);
            faceColor = tex2D(ColorMap,irisST.yz).rgb;
        }
        faceColor = lerp(faceColor,LensColor,fadeT);
        hitColor = lerp(missColor,faceColor,smoothstep(0.0h,GRADE,slice));
    }
}
```

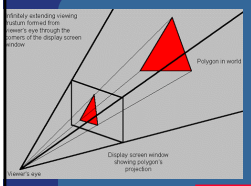

PixelShader 3.0

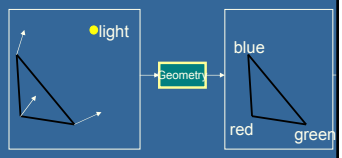
```
// if (-dir.z/|dir| > cos(PI/4)) t1 = zero
dp3 r6.w, r6, r6
rsq r6.w, r6.w          ← normalization
mad r0.w, -r6.z, r6.w, -CosPiOverFour
cmp r10.y, r0.w, Zero, r10.y

// set r10 to 0 if Disc <= 0
cmp r10.xy, -r7.w, Zero, r10

// compute r1 and r2 clipped
mad r1.xyz, r6, r10.x, r4          // IP0
mad r2.xyz, r6, r10.y, r4          // IP1

// project
rcp r11.w, r1.z
mad r1.xyz, r1, r11.w, NegZ        // P0
rcp r11.w, r2.z
mad r2.xyz, r2, r11.w, NegZ        // P1

// Compute area
texld r3, r1, ATan2Texture         // theta0
texld r4, r2, ATan2Texture         // theta1

crs r5.z,r1,r2                           // z =
abs r5.z,r5.z

mov r3.y, r4.x
texld r4, r3, SphAreaTexture       // lookup theta/PI
```

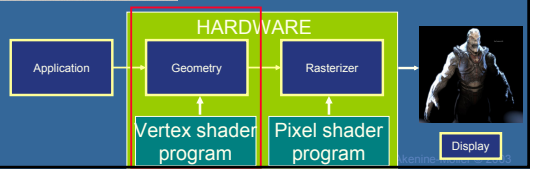
- Only floats
- Instructions operate on 1,2,3 or 4 components
  - x,y,z,w or
  - r,g,b,a
- Free Swizzling
- Only read from texture
- Only write to pixel (4-5 output buffers)

Hardware design

Geometry - per vertex:
- Lighting (colors)
- Screen space positions

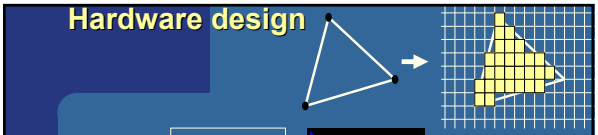


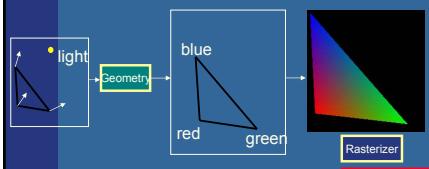




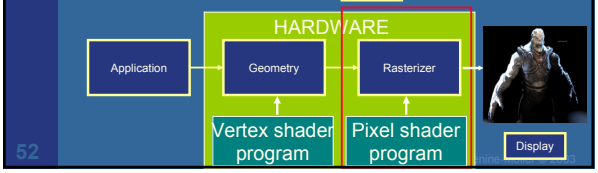
51

Hardware design





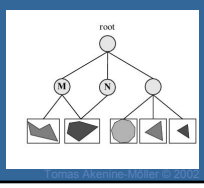
Rasterizer:
- Rasterization
- Textures
- Interpolated colors



52

Scene graphs

The scene graph is a hierarchical scene description with e.g.:
- Lights
- Textures
- Transforms
- And more



53