**OpenGL: Overview**

- Background on OpenGL
- Primitives
- The OpenGL Pipeline
- The OpenGL state machine
- Cameras, light and shading, textures
- Graphics data arrays
- Buffering, copying and filtering
- Animation and effects
- GLU and GLUT

---

**Background on OpenGL**

- Originally developed by SGI (Silicon Graphics Inc.).
- Controlled today by ‘ARB’.
- API for processing streams of 2D and 3D primitives:
  - Transform, Lighting, Shading, Texturing, Pixel ops.
- OpenGL is not intelligent – it’s not a scene-graph.
- OpenGL provides no support for the windowing system
  - GLU ‘layer’ provides windowing system independent support
- Discussion forums, documentation and examples:
  - [www.opengl.org](http://www.opengl.org)
  - [www.sgi.com/software/opengl](http://www.sgi.com/software/opengl)

---

**Design Goals**

- Real-Time: 30-120 frames/s
- Double buffering:
  - for smooth transitions
  - View one buffer, render to the other
  - 4 for stereo, of course.
- All code executed for every frame:
  - OpenGL remembers nothing, everything must be re-rendered
- Animation
  - Render all graphics for an image, wait for sync, swap buffers
- Version and Extensions:
  - `glGetString` (GL_VENDOR, GL_RENDERER, GL_VERSION, GL_EXTENSIONS)

---

**API conventions**

- Function names begin with `gl` and use mixed case:
  - `glBegin`, `glEnd`, `glFrontFace`
- Constants begin with `GL_` and use only upper case:
  - `GL_TRIANGLES`, `GL_BACK`, `GL_LIGHT0`
- Types begin with `GL` and use only lower case:
  - `GLubyte`, `GLshort`, `GLint`, `GLfloat`, ...

---

**API conventions: Function names show arguments**

- `glVertex{234}{sifd}[v]()`
  - `glVertex3l(GLint x,Glnt y,Glnt z)`
  - `glVertex3fv(GLfloat *v)`
- Argument types:
  - `GLbyte`, `GLshort`, `GLint`, `GLfloat`, `GLdouble`
  - unsigned byte (`ub`), short (`us`), int (`ui`)
- `v` indicates vector
Example function names

- `glVertex2s(1, 2);`
- `glVertex3d(1.0, 2.0, 3.142);`
- `glVertex4f(1.0, 2.0, 3.142, -6.284);`
- `GLdouble dvec[3] = {1.0, 2.0, 3.142};`
- `glVertex3dv(dvec);`

Primitives

- Points
  - `GL_POINTS`
- Lines
  - `GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP`
- Triangles
  - `GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN`
- Quadrilaterals and all other polygons:
  - `GL_QUADS, GL_QUAD_STRIP, GL_POLYGON`
- Ordering of vertices (corners) defines front & back
  - `-GL_CCW` (default)

GL Primitives - Triangles

```c
glBegin(GL_TRIANGLES);
  /* First Triangle */
  glVertex3f(1.0, 0.0, 0.0);
  glVertex3f(-0.5, 1.0, 0.5);
  glVertex3f(0.0, -0.5, -0.2);
  /* Second Triangle */
  glVertex3f(1.0, -0.4, 0.0);
  glVertex3f(0.0, 0.6, 0.0);
  glVertex3f(-0.6, 0.6, -0.2, 0.4);
glEnd();
```

Sample pseudoprogarm

```c
#include <stuff.h>

int main(int argc, char **argv)
{
  MakeGraphicsWindow(); /* Not done in OpenGL */
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
  glBegin(GL_POLYGON);
    glVertex3f(0.25, 0.25, 0.0);
    glVertex3f(0.75, 0.25, 0.0);
    glVertex3f(0.75, 0.75, 0.0);
    glVertex3f(0.25, 0.75, 0.0);
  glEnd();
  glFlush();
  UpdateWindowAndWaitForEvents(); /* Not GL's job */
  return 0;
}
```

Pipeline

- Display List
  - Stores vertex and pixel data for later processing
  - Can be stored in the graphics hardware for speed and reuse
  - Particularly good for matrix ops (transforms)

PIPELINE

- Pipeline
  - Per-Vertex Operations
  - Pixel Operations
  - Texture Assembly
  - Fragment Operations
  - Frame-buffer
• Evaluators
  • Handles parametric shapes:
    - Curves and surfaces
      - e.g., NURBS, Quadrics, Tesselators

  • Per-Vertex Operations and Primitive Assembly
    • Transforms to screen coordinates
    • Lighting, Texture coordinate processing
    • Clipping, Perspective Projection

  • Display List Evaluators
  • Per-Vertex Operations and Primitive Assembly
  • Pixel Operations
    • Rescaling and biasing of pixel data
    • Unpacking and repacking of pixel data
    • Filtering

  • Pixel Operations
  • Rasterization
    • Polygon, line and vertex information into raster fragments of the frame buffer (pixels...almost).

  • Texture Assembly
    • Building, storing and prioritizing texture objects

  • Per-Fragment Operations
    • Texturing, fog, alpha, depth and stencil tests, blending, dithering, masking, etc.
**THE GL STATE MACHINE**

- State machine - you define the state
  - Send parameters
  - Define controls
  - Define colours
  - Define textures
  - Define transforms
  - Send primitives
- Then execute the state

**CONTROLLING THE STATE**

- glEnable and glDisable
  - e.g.: GL_LIGHTING, GL_DEPTH_TEST
- glPushMatrix or glPopMatrix
  - glPushMatrix creates a copy of the current matrix on the stack – many deep?
  - glPopMatrix throws away the current one and recovers the previously pushed one
- Error control and feedback
  - glGetError, gluErrorString
  - glRenderMode, glPassThrough, glFeedbackBuffer

**CAMERA AND PROJECTION MATRIX**

- Orthograhic or perspective projection
  - glFrustum or glOrtho
  - glViewport
- Front and back clip planes (Near, Far)
- Define matrix mode:
  - glMatrixMode(GL_PROJECTION)
  - glMatrixMode(GL_MODELVIEW)
- Actual view set by the model view-matrix

**COLOUR**

- Real colour (RGBA) or colour index
  - RGBA mode is more general than index mode
  - Colour index mode reduces the number of bits per pixel
  - Special effects 'tricks' like index-animation
- Colours defined for polygon vertices
  - glColor or glIndex
  - Shading affects colour:
    - GL_FLAT: Constant colour across polygon
    - GL_SMOOTH: Interpolation across polygon
**Lights**
- OpenGL defines 8 light sources
  - `glEnable(GL_LIGHTING, GL_LIGHT0 ... GL_LIGHT7)
- Parameters: `glLight*(...)*` (Ambient, Diffuse, Specular, Position, Spot Direction, Spot Exponent, Spot Cutoff, Constant Att., Linear Att., Quadratic Att.)
- Model for the lighting: `glLightModel*(...)*` (Ambient, Local Viewer, Two Sided (bidirectional))
- Switch on lighting with `glEnable(GL_LIGHTING)`

**Materials**
- To set the colour properties of the illuminated object (front and back)
  - `glMaterial`
  - `glColorMaterial`
- Parameters:
  - Ambient, Diffuse, Specular, Shininess, Emission
  - Parameter channels can create combinations (ambient-diffuse)
- Changing the material is costly!
  - Group polygons with similar materials if possible

**Shading**
- Correct normal vectors are required
  - Normal magnitude is always 1!
  - Automatic normalization is available
    - `glEnable(GL_NORMALIZE)`
  - Can give bad performance!
  - Asymmetric scaling affects the normals
    - Requires normalization
- GL_FLAT or GL_SMOOTH shading
  - `glShadeModel`
  - Set the normals correctly
    - One per polygon or one per vertex as appropriate

**Textures**
- 1, 2 or 3 dimensional 'images'
  - `glTexImage1D, glTexImage2D, glTexImage3D`
  - Texture dimensions are always $2^n$ ($2^n + 2$ if has a border)
  - `glTexSubImage` replaces a part of a texture
    - Often much cheaper than replacing the whole thing

**Textures - Usual uses: 2D**

**Textures - Usual Uses: 1D**
Textures - Other uses

- Mipmaps
  - Multim (in Parvo - Many things in a small place)
  - Use multiple pre-processed textures to handle level of detail changes when objects are far away
  - Avoids flickering and ‘scintillation’
- Texturobjects – A management system to control textures
  - glGenTextures, glBindTexture, glDeleteTextures
  - Store many textures in an object, switch between them without changing texture object.
  - Fast and efficient
  - Mosaic, UV-mapping

Vertex Arrays

- Store vertex data in arrays
  - Coordinates, colours (RGBA or index), normals, texture coordinates, polygon 'edge flags'
  - glVertexPointer, glColorPointer, glIndexPointer, glNormalPointer, glTexCoordPointer, glEdgeFlagPointer
- Packed arrays (improved memory access)
  - glInterleavedArrays
- Render primitives from array data
  - glArrayElement, glDrawElements, glDrawArrays

Frame Buffer management

- Colour, Depth, Alpha, Stencil, Accumulation
  - glClear* - initialize the buffer
  - glDrawBuffer, glReadBuffer
  - glAlphaTest, glStencilTest, glStencilOp
  - glDepthFunc
  - glAccum - Accum, Load, Return, Add, Mult

Pixel Copying

- Copy pixel data:
  - between buffers
  - from buffer to main memory
- glReadPixels, glDrawPixels, glCopyPixels
- glPixelStore, glPixelTransfer, glPixelMap
- Other Pixel operations can be applied
  - glPixelZoom

Effects

- Multipass rendering
  - Render images several times with different settings and blending the images
  - e.g. transparency – order of operations is wrong
  - e.g. different lighting and texturing passes
- Shadows, Bump-mapping, Per-pixel lighting
- Transform-matrix for colour encoding
- Lots of different methods

PIPELINE 2

- Stream processors
  - Programmable
  - Flexible
  - Fast!

Extra datachannels
Loops, conditionals, mathematical functions
Indirect texturing
SIGGRAPH2002 – RayTracer in a graphics processor
GL UTILITIES (GLU)

- Included in Standard OpenGL
- Higher level graphics primitives
  - gluSphere, gluCylinder, gluDisk
- Parametric surfaces:
  - Tesselators, Quadrics, NURBS
  - gluNewTess, gluNewQuadric, gluNewNurbsRenderer
- Camera management
  - gluLookat, gluPerspective, gluUnProject

Sample Program 2

- Sample program earlier didn’t work
  - No window
  - No management of events
- glut can do these things for us
- The revised program looks like...

Sample Program 2.1

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

void init (void)
{
  /* select clearing colour */
  glClearColor (0.0, 0.0, 0.0, 0.0);

  /* initialize viewing values */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
  glMatrixMode(GL_MODELVIEW);

  /* draw white rectangle with corners at (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0) */
  glBegin(GL_POLYGON);
  glVertex3f (0.25, 0.25, 0.0);
  glVertex3f (0.75, 0.25, 0.0);
  glVertex3f (0.75, 0.75, 0.0);
  glVertex3f (0.25, 0.75, 0.0);
  glEnd();

  /* don't wait! Immediately start processing buffered OpenGL routines */
  glFlush();
}
```

Sample Program 2.2

```c
void display()
{
  /* clear all pixels – note, no call to glClearColor*/
  glClear (GL_COLOR_BUFFER_BIT);

  /* draw white rectangle with corners at (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0) */
  glColor3f (1.0, 1.0, 1.0);
  glBegin(GL_POLYGON);
  glVertex3f (0.25, 0.25, 0.0);
  glVertex3f (0.75, 0.25, 0.0);
  glVertex3f (0.75, 0.75, 0.0);
  glVertex3f (0.25, 0.75, 0.0);
  glEnd();

  /* don't wait! Immediately start processing buffered OpenGL routines */
  glFlush();
}
```
Sample Program 2.3

/* Set initial window params & display mode. Open window with "hello" in * title bar. Call init routines, register callback to display graphics. Enter * main loop */

int main(argc, argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(250, 250);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("hello");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return(0);
}

Sample Program 2 - compilation

cc -o hello hello.c \
-I/usr/include/.../glut/include \ 
-L/usr/lib/.../glut/lib \ 
-lglut -lGLU -lGL -lXmu -lXext -lX11

So, now we know...

- Can Write basic GL programs in C
- Can create 3D objects from primitives
- Can colour and texture them
- Can interact with them through the main loop function

A Hardware implementation of OpenGL

- The SGI Onyx Infinite Reality 2 graphics Pipe
  - Just one example
  - There are many others
  - Your own PC probably has one!
- OpenGL lends itself well to hardware rendering
  - The state machine!

Onyx IR2 - Features

- 2.62 Million Pixels/pipe (10.48 Million/system)
- Multisampling and antialiasing
- CLIP mapping
- Projective textures
- Shadow Maps
- Convolution
- Dynamic Video Resolution (DVR)
- Genlock (generator locking device)
- Swap Synchronization for multiple pipes
Onyx IR2 - Architecture

- Graphics 'pipe' made up of a mix of:
  - Transform manager Board(s)
  - Raster Manager Board(s)
  - Display Generator Board(s)

Transform Manager Board (2 or 4 TE’s per board)

Raster Manager Board (1, 2 or 4 per pipe)

Display Generator Board (Option for 8 channels)

OpenGL on IR2

- Entire OpenGL pipeline is Hardware!
- Each pipe can accept up to 300MB/s of OpenGL commands
- TM system can process 11M Δ/s
- Pixel Generator can rasterize 12M Δ/s
- Fragment processor can produce 800M textured pixels/s