OpenGL, GLUT, CUDA, OpenCL, OpenCV, PointClouds, GLUT, and Qt

CS334 Spring 2025

Daniel G. Aliaga
Department of Computer Science
Purdue University

Computer Graphics Pipeline

Geometric Primitives

Modeling Transformation	Transform into 3D world coordinate system
Lighting	Simulate illumination and reflectance
Viewing Transformation	Transform into 3D camera coordinate system
Clipping	Clip primitives outside camera's view
Projection Transformation	Transform into 2D camera coordinate system
Scan Conversion	Draw pixels (incl. texturing, hidden surface…)

Image

OpenGL

- Software interface to graphics hardware
- ~150 distinct commands
- Hardware-independent and widely supported
  - To achieve this, no windowing tasks are included
- GLU (Graphics Library Utilities)
  - Provides some higher-level modeling features such as curved surfaces, objects, etc.
- Open Inventor (old)
  - A higher-level object-oriented software package

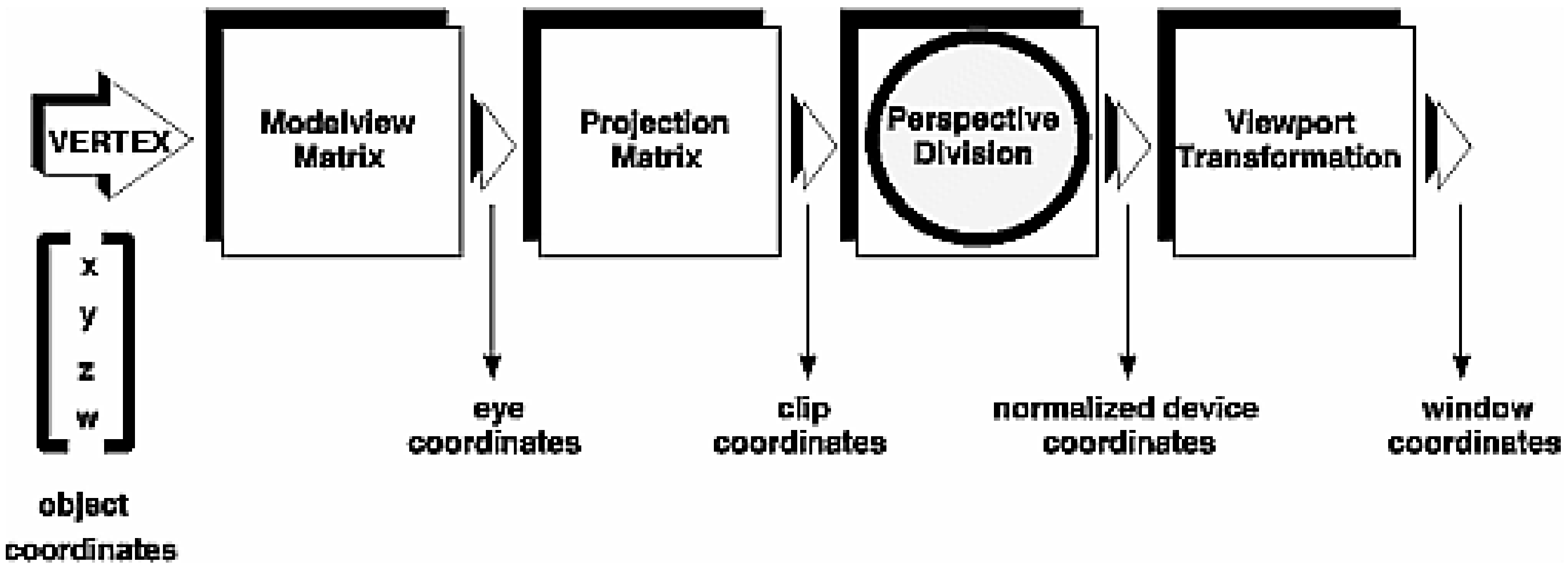
OpenGL Online

- Current version is: ~4.5
- Website
  - http://www.opengl.org
- Books
  - Programming Guide (“Red book”)
  - Reference Manual (“Blue book”)

OpenGL

- Rendering parameters
  - Lighting, shading, lots of little details…
- Texture information
  - Texture data, mapping strategies
- Matrix transformations
  - Projection
  - Model view
  - (Texture)
  - (Color)

Matrix Transformations



Matrix Transformations

- Each of modelview and projection matrix is a 4x4 matrix
- OpenGL functions
  - glMatrixMode(…)
  - glLoadIdentity(…)
  - glLoadMatrixf(…)
  - glMultMatrix(…)
  - glTranslate(…)
  - glScale(…)
  - glRotate(…)

  - glPushMatrix()
  - glPopMatrix()

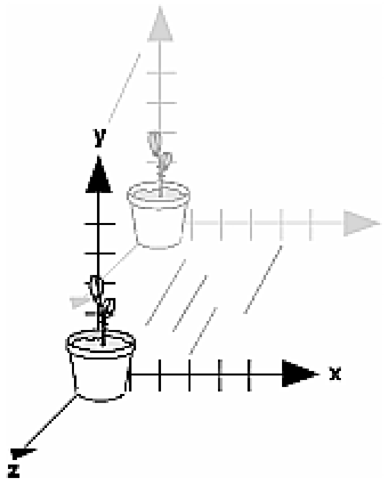
Matrix Transformations

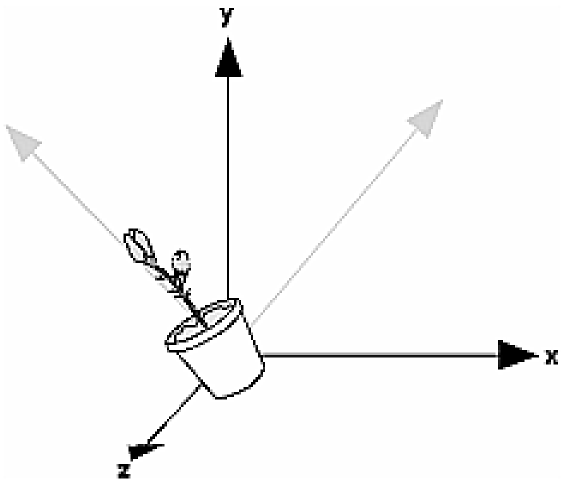
```
{
    …
    …
    …
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glMultMatrixf(N); /* apply transformation */
    glMultMatrixf(M); /* apply transformation M */
    glMultMatrixf(L); /* apply transformation L */
    glBegin(GL_POINTS);
     glVertex3f(v); /* draw transformed vertex v */
    glEnd();
    …
    …
    …
}
```

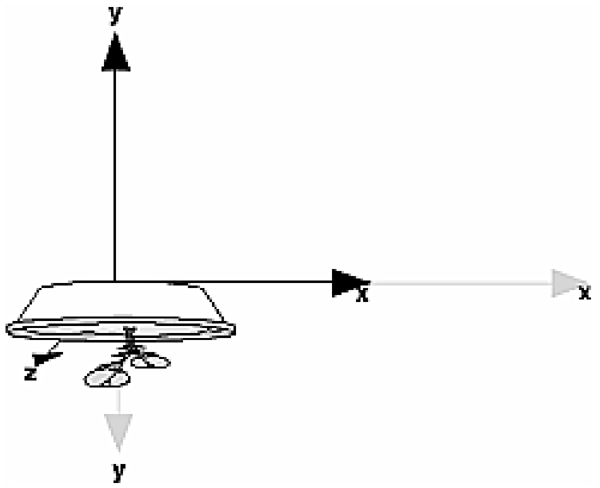
= draw transformed point “N(M(Lv))”

Modelview Transformations

`glRotatef(45,0,0,1)`



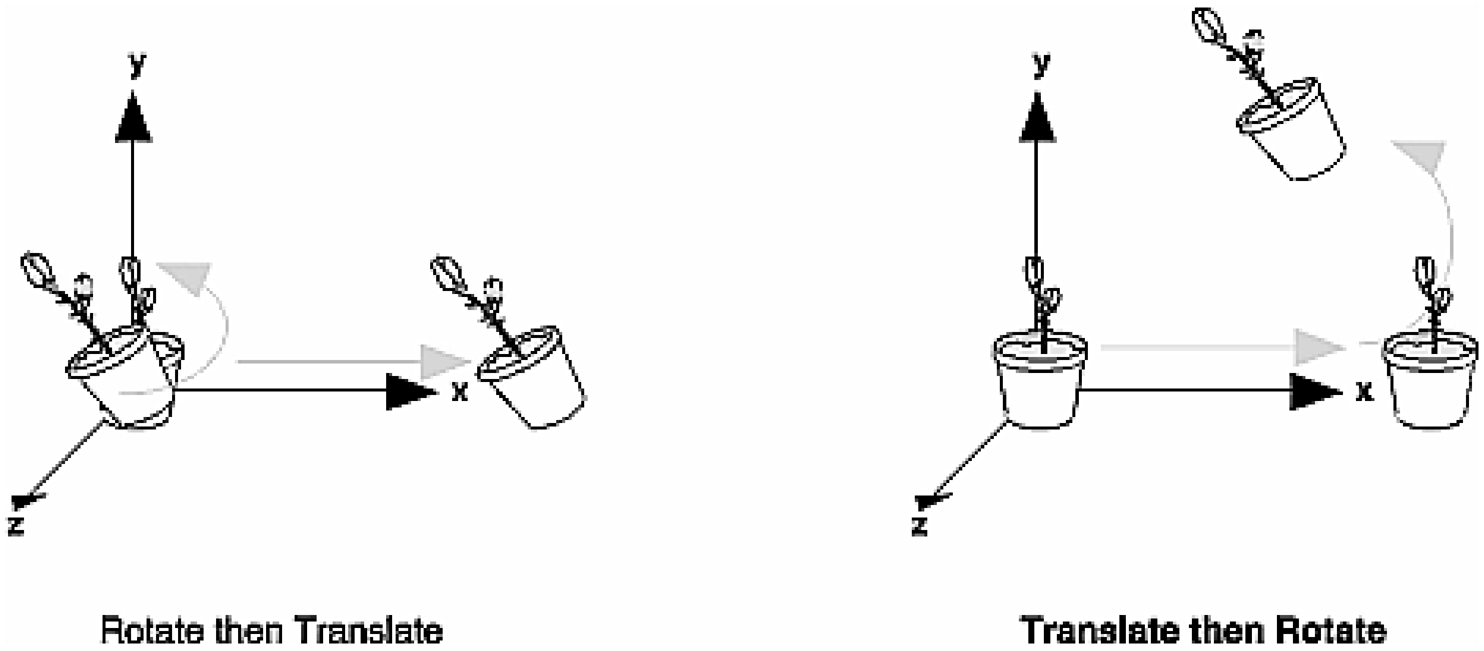




`glTranslate3f(tx,ty,tz)`

`glScalef(2,-0.5,1.0)`

Modelview Transformations



```
glRotatef(d,rx,ry,rz);
glTranslate3f(tx,ty,tz);
```

```
glTranslate3f(tx,ty,tz);
glRotatef(d,rx,ry,rz);
```

Simple OpenGL Program

```
{
   <Initialize OpenGL state>

   <Load and define textures>

   <Specify lights and shading parameters>

   <Load projection matrix>

   For each frame

       <Load model view matrix>
       <Draw primitives>

   End frame
}
```

Simple OpenGL Program

```
#include <GL/gl.h>
main()
{
   InitializeAWindowPlease();
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslate3f(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();
   UpdateTheWindowAndCheckForEvents();
}
```

GLUT/FreeGLUT

- = Graphics Library Utility Toolkit
  - Adds functionality such as windowing operations to OpenGL
- Event-based callback interface
  - Display callback
  - Resize callback
  - Idle callback
  - Keyboard callback
  - Mouse movement callback
  - Mouse button callback

Simple OpenGL + GLUT Program

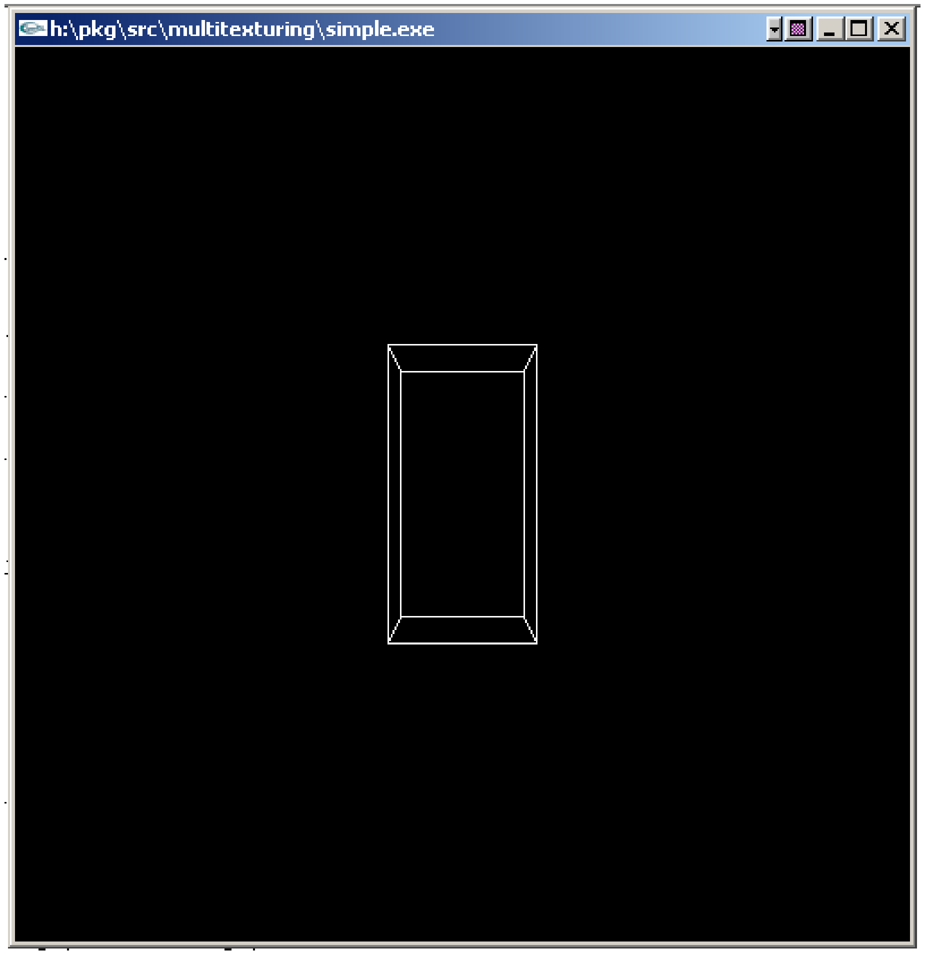
```
#include <…>

DisplayCallback()
{
    <Clear window>
    <Load Projection matrix>
    <Load Modelview matrix>
    <Draw primitives>
    (<Swap buffers>)
}

IdleCallback()
{
    <Do some computations>
    <Maybe force a window refresh>
}

KeyCallback()
{
    <Handle key presses>
}
```

```
KeyCallback()
{
    <Handle key presses>
}

MouseMovementCallback
{
    <Handle mouse movement>
}

MouseButtonsCallback
{
    <Handle mouse buttons>
}

Main()
{
    <Initialize GLUT and callbacks>
    <Create a window>
    <Initialize OpenGL state>

    <Enter main event loop>
}
```

Simple OpenGL + GLUT Program

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

void init(void)
{
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (1.0, 1.0, 1.0);
  glLoadIdentity ();
  gluLookAt (0, 0, 5, 0, 0, 0, 0, 1, 0);
  glScalef (1.0, 2.0, 1.0);
  glutWireCube (1.0);
  glFlush ();
}

void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
  glMatrixMode (GL_MODELVIEW);
}

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
  return 0;
}
```

Simple OpenGL + GLUT Program



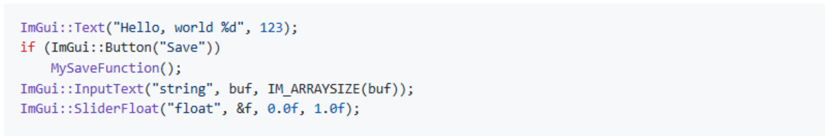
Example Program with Lighting

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

void init(void)
{
  GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
  GLfloat mat_shininess[] = { 50.0 };
  GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_SMOOTH);

  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_DEPTH_TEST);
}
void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glutSolidSphere (1.0, 20, 16);
  glFlush ();
}
```

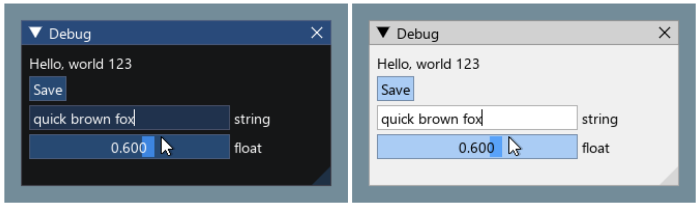
```
void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  if (w <= h)
    glOrtho (-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,
      1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
  else
    glOrtho (-1.5*(GLfloat)w/(GLfloat)h,
      1.5*(GLfloat)w/(GLfloat)h, -1.5, 1.5, -10.0, 10.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB |
       GLUT_DEPTH);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
  return 0;
}
```

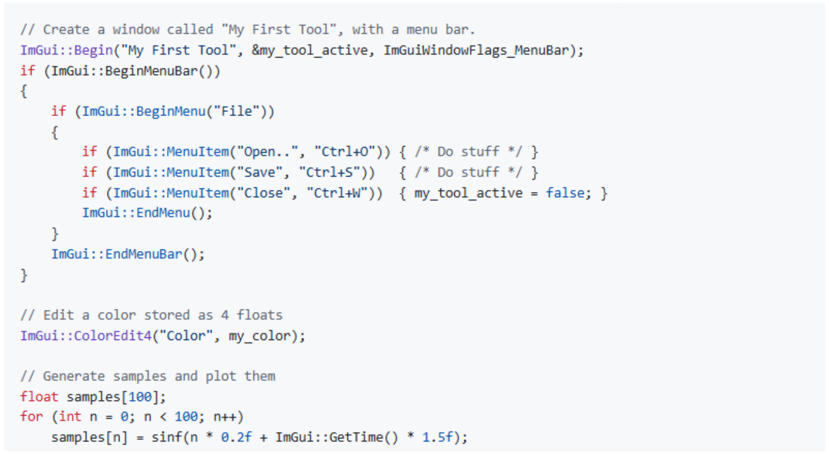
imGUI

- https://github.com/ocornut/imgui



```
ImGui::Text("Hello, world %d", 123);
if (ImGui::Button("Save"))
    MySaveFunction();
ImGui::InputText("string", buf, IM_ARRAYSIZE(buf));
ImGui::SliderFloat("float", &f, 0.0f, 1.0f);
```





```
// Create a window called "My First Tool", with a menu bar.
ImGui::Begin("My First Tool", &my_tool_active, ImGuiWindowFlags_MenuBar);
if (ImGui::BeginMenuBar())
{
    if (ImGui::BeginMenu("File"))
    {
        if (ImGui::MenuItem("Open..", "Ctrl+O")) { /* Do stuff */ }
        if (ImGui::MenuItem("Save", "Ctrl+S"))   { /* Do stuff */ }
        if (ImGui::MenuItem("Close", "Ctrl+W"))  { my_tool_active = false; }
        ImGui::EndMenu();
    }
    ImGui::EndMenuBar();
}

// Edit a color stored as 4 floats
ImGui::ColorEdit4("Color", my_color);

// Generate samples and plot them
float samples[100];
for (int n = 0; n < 100; n++)
    samples[n] = sinf(n * 0.2f + ImGui::GetTime() * 1.5f);
```

CUDA and OpenCL

- NVIDIA defined “CUDA” (new)
  – Compute Unified Device Architecture
  – http://www.nvidia.com/object/cuda_home.html#

- Khrono’s group defined “OpenCL” (newer)
  – Open Standard for Parallel Programming of Heterogeneous Systems
  – http://www.khronos.org/opencl/

CUDA Example

- Rotate a 2D image by an angle
  - On the CPU (PC)
    - simple-tex.pdf
  - On the GPU (graphics card)
    - simple-tex-kernel.pdf

OpenCL Example

- Compute a Fast Fourier Transform
  - On the CPU (PC)
    - cl-cpu.pdf
  - On the GPU (graphics card)
    - cl-gpu.pdf

OpenCV

- A library for computer-vision related software
- Derived from research work and high-performance code from Intel
- http://opencv.willowgarage.com/wiki/

Point Clouds

- For 3D point cloud processing and rendering
  - http://www.pointclouds.org

Microsoft XNA

- A Visual Studio programming environment to create games for Windows Phone, Xbox 360, and Windows-based computers
  - http://www.microsoft.com/download/en/details.aspx?id=23714

Unity 3D

- Unity is a game development ecosystem: a powerful rendering engine fully integrated with a complete set of intuitive tools and rapid workflows to create interactive 3D content; easy multiplatform publishing; thousands of quality, ready-made assets in the Asset Store and a knowledge-sharing Community.
  - http://unity3d.com/unity/

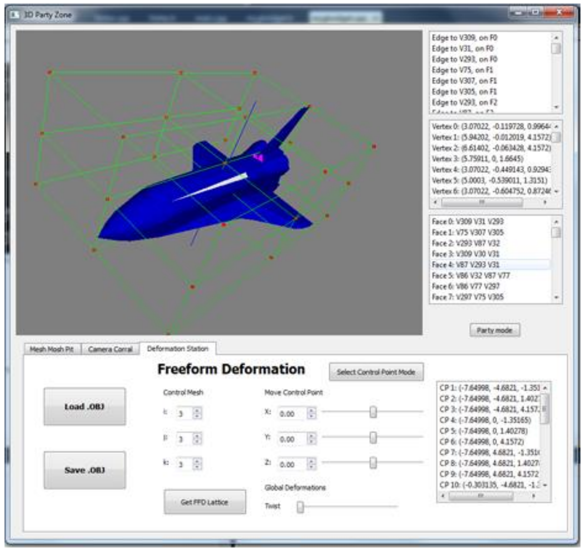
G3D

- The **G3D Innovation Engine** is
  - an open-source commercial-grade C++ 3D engine
  - used in commercial games, research papers, military simulators, and university courses.
  - supports hardware accelerated real-time rendering, off-line rendering like ray tracing, and general purpose computation on GPUs.
- http://g3d.sourceforge.net

Qt

- Qt is a cross-platform application framework, including OpenGL support, that can be run on various software and hardware platforms

www.qt.io



Linear Algebra

- Why do we need it?
  - Modeling transformation
    - Move “objects” into place relative to a world origin
  - Viewing transformation
    - Move “objects” into place relative to camera
  - Perspective transformation
    - Project “objects” onto image plane