What is Computer Graphics?
Objectives of this course
Administrivia
Introduction to the OpenGL pipeline
Computer Graphics

One of the “core” computer science disciplines

- Algorithms and Theory
- Artificial Intelligence
- Computer Architecture
- Computer Graphics and Visualization
- Computer Security
- Computer Systems
- Databases
- Networks
- Programming Languages
- Software Engineering
Why Computer Graphics?

- Movies
- Games
- CAD-CAM
- Simulation
- Virtual Reality
- Visualization
- Medical Imaging
- other?
Course Objectives

- Learn the fundamentals of computer graphics algorithms
- Learn to program 3D interactive graphical applications using OpenGL
- Gain an understanding of graphics APIs and the graphics hardware
- Learn advanced features of OpenGL, such as lighting, textures, shading, and so on.
Prerequisites

- Programming skills in C/C++ family of languages
- Basic concepts in linear algebra & matrices
- Helps to have basic knowledge of data structures
- JavaScript a plus
Course Overview

- Theory: Computer Graphics disciplines:
  - Modeling: how to represent objects
  - Animation: how to control and represent motion
  - Rendering: how to create images of objects
  - Image Processing: how to edit images
- Practice: OpenGL/WebGL graphics library

Not in this course:
- Human-Computer Interaction
- Graphic Design
- DirectX API
Computer Graphics Disciplines

Rendering

Geometry (Modeling)

Animation

Image Processing

Source: Jensen

Source: Baraff and Witkin

Source: Botsch et al.

Source: Durand
Main Computer Graphics event in the world
Academia and Industry

www.siggraph.org

SIGGRAPH 2015 will be held in Los Angeles, August 9-13 (s2015.siggraph.org)

✦ SIGGRAPH 2015 Student Volunteer applications are now open
✦ ACM Student Research Competition
Administrivia

- course webpage: http://www.clarkson.edu/~jets/cs452
- contact info:
  office: SC375 (or Applied CS labs, SC334-336)
  phone: 268-2377,
  email: jets@clarkson.edu
Textbook

- **Required:**
Other resources


Grading Policy

- 25% 1 Midterm Exam (tentatively 3/10)
- 30% Final Exam
- 20% Labs, Homework & Quizzes
- 25% Programming Projects
What is OpenGL?

- OpenGL is a computer graphics rendering application programming interface (API)
  - With it, you can create interactive applications that render high-quality color images composed of 2D and 3D geometric objects and images
  - It forms the basis of many interactive applications that include 3D graphics
  - By using OpenGL, the graphics part of your application can be
  - operating system independent
  - window system independent
OpenGL ES

- OpenGL ES 2.0
  - Designed for embedded and hand-held devices such as cell phones
  - Based on OpenGL 3.1
  - Shader based
What is WebGL?

- WebGL: JavaScript implementation of OpenGL ES 2.0
  - runs in all recent browsers (Chrome, Firefox, IE, Safari)
    - operating system independent
    - window system independent
  - application can be located on a remote server
  - rendering is done within the browser using local hardware
- uses HTML5 canvas element
- integrates with standard web packages and apps
  - CSS
  - jQuery
Why WebGL

- Runs across multiple devices including many smart phones
- No system dependencies
- Uses local resources
- Integrates with other web applications
High-Profile WebGL Uses
In modern versions of OpenGL (version 3.1 and up), applications are entirely shader based.
Generally speaking, data flows from your application through the GPU to the framebuffer.
Your application will provide vertices, which are collections of data that are composed to form geometric objects.
The vertex processing stage uses a vertex shader to process each vertex, doing any computations necessary to determine where in the frame buffer each piece of geometry should go.
After all the vertices are processed, the rasterizer determines which pixels in the frame buffer are affected by the geometry, and for each pixel, the fragment processing stage uses a fragment shader to determine the final color of the pixel.
In your OpenGL applications, you’ll usually need to do the following tasks:

✦ specify the vertices for your geometry
✦ load vertex and fragment shaders (and others)
✦ engage the OpenGL pipeline for processing
Modern OpenGL programs essentially do the following steps:

1. Load and create OpenGL shader programs from shader source programs you create
2. Load the data for your objects into OpenGL’s memory. You do this by creating buffer objects and loading data into them.
3. Continuing, OpenGL needs to be told how to interpret the data in your buffer objects and associate that data with variables that you’ll use in your shaders. We call this shader plumbing.
4. Finally, with your data initialized and shaders set up, you’ll render your objects.
OpenGL applications need a place to render into
-usually this is an on-screen window
Need to communicate with the native windowing system – but each windowing system interface is different
We use GLUT (more specifically, freeglut)
-similar, open-source library that works everywhere
-handles all windowing operations
Evolution of the OpenGL Pipeline
In the Beginning …

- OpenGL 1.0 was released on July 1\textsuperscript{st}, 1994
- Its pipeline was entirely \textit{fixed-function}
  - the only operations available were fixed by the implementation

- The pipeline evolved
  - but remained based on fixed-function operation through OpenGL versions 1.1 through 2.0 (Sept. 2004)
Beginnings of The Programmable Pipeline

- OpenGL 2.0 (officially) added programmable shaders
  - *vertex shading* augmented the fixed-function transform and lighting stage
  - *fragment shading* augmented the fragment coloring stage
- However, the fixed-function pipeline was still available
An Evolutionary Change

- OpenGL 3.0 introduced the *deprecation model* – the method used to remove features from OpenGL
- The pipeline remained the same until OpenGL 3.1 (released March 24th, 2009)
- Introduced a change in how OpenGL contexts are used

<table>
<thead>
<tr>
<th>Context Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full</strong></td>
<td><em>Includes all features (including those marked deprecated)</em> available in the current version of OpenGL</td>
</tr>
<tr>
<td><strong>Forward Compatible</strong></td>
<td><em>Includes all non-deprecated features (i.e., creates a context that would be similar to the next version of OpenGL)</em></td>
</tr>
</tbody>
</table>
OpenGL ES and WebGL

• OpenGL ES 2.0
  – Designed for embedded and hand-held devices such as cell phones
  – Based on OpenGL 3.1
  – Shader based

• WebGL
  – JavaScript implementation of ES 2.0
  – Runs on most recent browsers
WebGL Application Development
Simplified Pipeline Model

Application → GPU Data Flow → Framebuffer

- **Vertices**
  - Vertex Processing
  - Vertex Shader

- **Vertices**
  - Rasterizer

- **Fragments**
  - Fragment Processing
  - Fragment Shader

- **Pixels**
  - Framebuffer
• All WebGL programs must do the following:
  – Set up canvas to render onto
  – Generate data in application
  – Create shader programs
  – Create buffer objects and load data into them
  – “Connect” data locations with shader variables
  – Render
Application Framework

- WebGL applications need a place to render into
  - HTML5 Canvas element
- We can put all code into a single HTML file
- We prefer to put setup in an HTML file and application in a separate JavaScript file
  - HTML file includes shaders
  - HTML file reads in utilities and application
A Really Simple Example

- Generate one red triangle
- Has all the elements of a more complex application
  - vertex shader
  - fragment shader
  - HTML canvas

- www.cs.unm.edu/~angel/WebGL
<!DOCTYPE html>
<html>
<head>
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
void main()
{
   gl_Position = vPosition;
}
</script>
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
void main()
{
   gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
</script>
</head>
</html>
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="triangle.js"></script>
</head>
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
var gl;
var points;

window.onload = function init()
{
    var canvas = document.getElementById( "gl-canvas" );
    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" );
}

var vertices = new Float32Array([-1, -1, 0, 1, 1, -1]);

    // Configure WebGL

    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl clearColor( 1.0, 1.0, 1.0, 1.0 );
// Load shaders and initialize attribute buffers

var program = initShaders( gl, "vertex-shader", "fragment-shader" );

gl.useProgram( program );

// Load the data into the GPU

var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW );
// Associate out shader variables with our data buffer

    var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );
    render();
};

function render()
{
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.TRIANGLES, 0, 3 );
}
For next time...

- Read Chapter 1 of Angel & Shreiner