CSE 167 outline – Fall 2003 – Instructor: Sam Buss

These are notes to guide my lectures and to let students review an outline of the lectures. They represent my ***plans*** for the lectures --- actual lectures may be a little different.

9/25/03 **Preliminaries.**

Bring good chalk, textbooks, some overhead transparencies with old projects.
Announce self, office (APM 6210), Email: sbuss@ucsd.edu. (Best way to reach me.)
Initial office hours: Friday 9/26 at 9:30 and Monday 9/29 at 10:30.
Regular office hours to be announced next week.
TA's are Craig Donner, Peter Schwer and Diem Vu.
Course web page. http://math.ucsd.edu/~sbuss/CourseWeb/CSE167_2003F/
Textbook web page: http://math.ucsd.edu/~sbuss/MathCG/
Course content. Basics of OpenGL, drawing primitives, transformations in 2D and 3D, affine transformations, perspective, interpolation and averaging, color, materials, texture mapping, Phong lighting, Gouraud and Phong shading, Bézier curves.
Course textbook. One required, one recommended. Bring and show off.
Course requirements.

> Projects (probably 7 projects). Final project is open ended. Must use PCs in the APM 2444 lab to demo your projects for grading. Lab door code: 0593110
> Midterm and final exam.
> Homework assignments. Not sure if they will be collected and graded.
> Possibly quizzes (held in class, preannounced).
> Grading approximately 50% projects, 50% exams, possibly homework or quizzes.

Samples from student final projects from earlier years.
OpenGL and GLUT as platform-independent API for 3D computer graphics.
Prerequisites: Programming skills in C, C++ or Java. Math 20C & F.
Math content: calculus and linear algebra, but not to worry, can relearn as we go along.
About me: Professor of math and computer science. Long-term programmer, game development consultant.
This course has a successor: Math 155B in the winter quarter.
First project online already. Hand out course account slips at end of class.

Starting the course.

Display models. Pixels (rectangular array of). Pixels display color (RGB).

Drawing points, drawing lines. Drawing surface patches (this final one is the real focus of "3D" computer graphics). A solid figure is generally rendered by drawing its surface.

Example of drawing a sphere and a house. Conventions on x,y axes in 2D and on x,y,z axes in three dimensions.

The OpenGL commands for (a) points, (b) lines etc., (c) triangles, quads and polygons.

```
glBegin( ----- );
….
glVertex*(…);     // One command per vertex or point specified.
….
glEnd( );
```

The following are possible parameters to glBegin(…);

```
GL_POINTS
GL_LINES
GL_LINE_STRIP
GL_LINE_LOOP
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
GL_QUADS
GL_QUAD_STRIP
GL_POLYGON
```

Examples of how these various drawing modes work (hand-draw on blackboard).

Simple uses of glColor3f(r,g,b) to render solid colors.

Animation. Buffers. Double buffering.

Reading assignment for above topics.
Buss: pp. 1-16. (Chapter 1.)
The Red Book, pp. 1-9, 14-48. (Chapter 1, skipping the "Rendering Pipeline" section; and the first part of Chapter 2 up through the "Describing Points, Lines and Polygons" section.)
From the textbook web site, check out the **SimpleDraw** and the **SimpleAnim** programs. See if you can understand all of the code in these two programs.

Transformations in 2D (Mathematical definitions)

The rendering pipeline (simplified, four stage) – see Buss, page 18.

9/30/03 Definition of a **transformation.**
Mathematical definition of a **linear transformation** in 2D.
Examples. Pictures of how example linear transformations act on a "F".
Definition. A translation, $T_{\mathbf{u}}$. Examples.
Composition of transformations.
Inverse transformation.
Definition of an **affine transformation**.
Every affine transformation is uniquely expressible as composition of translation and linear transformation.

2x2 matrices represent a linear transformation.

Matrix represents a linear transformation.
Definition of **i** and **j.**
Proof that every linear transformation is represented by a 2x2 matrix.
Example.

Special conventions on vectors. <x, y> is a column vector. (x, y) is a row vector.
Transposes of matrices and vectors.

Example of rotation.
General rotations in 2D (around the origin): R_θ

Rigid transformations.
Rigid, orientation-preserving transformations.
Orthonormal matrices. Definition of. They represent rigid transformations.

Generalized rotations (rotations about a point other than the origin). R_θ^{u}

Thm: Every rigid, orientation-preserving linear transformation is a rotation.
Thm: Every rigid, orientation-preserving affine transformation is either a translation or a generalized rotation.

Reading assignment for the above topics.
Buss, pp. 17-26. (Sections II.1.1 - II.1.3).

10/2/03

Homogeneous coordinates in 2-space

The meaning of homogenous coordinates <x,y,u>. It represents the point < x/u, y/u > in 2-space (for u non-zero).
Several examples.
Points at infinity are represented with triples that have u equal to zero.

Matrix representations of transformations that act on homogeneous coordinates.
3x3 matrix represents an affine transformation in 2-space.
Examples.
For now, we are mostly interested in matrices that have bottom row (0 0 1).

Thm: The composition of two affine transformations is affine.

Reading assignment for the above topics: Buss, pp. 26-28. (Sections II.1.4 and II.1.5).

(Psuedo) OpenGL commands for 2D transformations

```
glMatrixMode( GL_MODEL_VIEW );
glLoadIdentity( );
pglTranslatef( x, y);              // Not a real OpenGL command
```

```
pglRotate( theta );                // Not a real OpenGL command
glPushMatrix();
glPopMatrix();
```

The **drawThreeDots** example.

Two ways of viewing transformations's actions: (1) acting on objects, moving them around the *xy*-plane, and (2) acting on a local coordinate system.

Reading assignment for above topics: Buss, pp. 28-32. (Sections II.1.6, and II.1.7).

Transformations in 3-space (Mathematical definitions).

Most of the definitions and concepts for transformations in 2-space generalize to 3-space.
10/7/03 Note the unusual placement of the *x, y, z* axes!
Redefinition of linear transformation, translation, affine transformation.
Homogeneous coordinates for points in 3-space <x,y,z,u>.
Matrix representations. A 3x3 matrix represents a linear transformation in 3-space.
A 4x4 matrix represents an affine transformation in 3-space based on homogenous coordinates.
Rigid transformations. Orientation-preserving transformations.

Rotations in 3-space. $R_{\theta,\mathbf{u}}$ – rotate around **u**, direction given by right-hand rule.

Reading for above: Buss, pp. 34-36 (Section II.2.1).

(Actual!) OpenGL commands for 3D transformations

```
glMatrixMode( GL_MODEL_VIEW );
glLoadIdentity( );
glTranslate3f( x, y, z );
glRotatef ( theta, x, y, z );
glScalef( a, b, c );
glLoadMatrixf( float * );
glMultMatrixf( float * );

glPushMatrix( );
glPopMatrix( );
```

Solar System example.
Discuss Project #1. (Assignment available on the web.)

Reading for above: Buss, pp. 36-40 (Section II.2.2).

The Rotation Matrix formula

Dot product and cross products.
Projection onto a line. Projection onto a plane.
Derivation of the formula for 4x4 matrix that represents a rotation in 3-space.

Euler's Theorem. Every rigid, orientation-preserving linear transformation is a rotation around a axis through the origin.

Screw motions (glide rotations).

Reading for above: Buss, pp 41-45. (Sections II.2.3, II.2.4.)

10/9/03 OpenGL – Simple Programs – Setting up an OpenGL program

(Guest lecture by Craig Donner)

Examples from SimpleDraw and SimpleAnim (available from Buss book web site).

Window Creation
Window Resizing
Depth buffer – requesting and enabling
glutSwapBuffers
Callbacks – Event-driven style programing
- Display Function, glutDisplayFunc
- Idle function, glutIdleFunc
- glutReshapeFunc
- Keyboard and special characters. glutKeyboardFunc, glutSpecialFunc

Modelview and Projection matrices – when they are called.
Double buffering and animation.
- glutSwapBuffers, glutSwapBuffers

10/14/03 Projective Geometry

In 2-space (R^2):
Points at infinity.
A point at infinity is represented by homogeneous coordinates <x, y, 0>. It is viewed as being out at the "end" of the line with slope y/x. The same point lies at the other end of the line, so the line effectively "wraps around" at infinity.
Two lines with the same slope intersect at the same point at infinity.
The points at infinity form the "line at infinity".

Similar constructions apply to three dimensional projective space.

Reading for above: Buss, pp. 32-34 and 45-46. (Sections II.1.8 and II.2.5.)

Viewing Transformations in OpenGL

Orthographic projections and perspective projections.
Advantages of orthographic versus perspective projections.
OpenGL uses viewing transformations to map the visible part of the scene into a cube.
This cube is a 2x2x2 cube centered at the origin.
The cube is then projected orthographically on to the display, depth or distance information is used to perform hidden surface computations.

To use viewing transformations in OpenGL:

- Set the viewer at the origin, looking down the negative z-axis. (Otherwise lighting and materials will not work properly.)
- Use an **glOrtho()** or **gluPerspective()** command to set the viewer's field of view.
- This includes a **near clipping plane** and a **far clipping plane**.
- Vertices are transformed by the composition of the GL_PROJECTION matrix and the GL_MODELVIEW_MATRIX.

glOrtho (left, right, bottom, top, near, far);

gluPerspective(theta, aspectRatio, near, far);

aspectRatio is the ratio of the width of the viewed field to its height.

glFrustrum(left, right, bottom, top, near, far);

glFrustrum( ) is more general than gluPerspective( ). gluPerspective( ) is usually easier and good enough for most applications.

gluLookAt(eye-x, eye-y, eye-z, center-x, center-y, center-z, up-x, up-y, up-z);

gluLookAt should only be used when the GL_MODELVIEW matrix is the currently active matrix. The glOrtho, gluPerspective, glFrustrum commands are usually given when the GL_PROJECTION matrix is the currently active matrix.

Rendering principles

The mapping of viewable region into a 2x2x2 cube, centered at the origin.
How vertices of a triangle map to pixels on the screen.
General discussion of how the interior of the triangle is filled in based on the pixels containing the pixels.
Colors and depth values of points interior to the triangle are averaged from the vertices.

Readings for the above: Various parts of Buss, pp. 46-58 (Section II.3).

10/16/03

Viewing Transformations – the mathematical theory

The 4x4 matrix for orthographic projections

The 4x4 matrix for perspective projections.
This matrix does not have “0 0 0 1” as its fourth row.
The depth or distance to a point used for hidden surfaces..
How depth buffering works.
Comparison with the painter’s algorithm and geometric algorithms.

Problems with naïve choices for the distance function.
Interpolation is used to draw lines and triangles and quads from their vertices.
Depth values are interpolated also (along with color, etc.)

The pseudo-distance function *psuedodist*(z) = A+B/z.
The monotonicity of the pseudo-distance function.
Calculation of A and B: need *pseudodist*(near) = 1, and *psuedodist*(far) = -1
The 4x4 matrix representation of perspective transformations with the pseudo-distance function.
Using this pseudo-distance function causes line to map to lines, hence the depth buffer problems do not occur.
Near and far clipping planes must be chosen properly.

Shadows

Using perspective transformations for shadows onto a flat surface.

Following commands can be used to prevent z-fighting of shadows laying exactly on top of the flat surface.

```
glPolygonOffset (1.0, 1.0);
glEnable( GL_POLYGON_OFFSET_FILL );
```

These are also used for things like placing flat images directly on flat surface, e.g., paintings on a wall.

Readings for the two topics: Remaining parts of Buss, pp. 46-58 (Section II.3).

Bresenham algorithm

Interpolation (that is, averaging) revisited. Let a be a real number between 0 and 1. Let $\mathbf{x}$ and $\mathbf{y}$ be points in 3-space. The point which is fraction a of the way from $\mathbf{x}$ to $\mathbf{y}$ is given by the formula

$$(1-a)\mathbf{x} + a\mathbf{y}.$$

This can be equivalently written as $\mathbf{x} + a(\mathbf{y}-\mathbf{x})$ (which may be easier to understand).

This interpolation formula is used for averaging colors and depth values along a line, and similar formulas are used to average values inside a triangle.

Given a line of slope less than 1 and greater than -1 which is being on a screen, the same interpolation formula is used to compute y coordinates of pixels on the line. (See picture on page 60 of the textbook.) For lines of slope more than 1 or less than -1, the roles of x and y are interchanged.

The Bresenham algorithm uses this idea to draw lines. Code for a floating point version of the Bresenham algorithm, and for a much faster integer based version are shown in pages 62-63 of the book.

Similar constructions are used to draw triangles.

Some general principles: if two triangles share an edge, then pixels that are drawn for the triangles should have the property that (a) no pixel belongs to both triangles, and (b) there are no gaps of pixels left undrawn between the two triangles.

Cautions

You should be careful to make adjacent triangles or quadrangles share exactly the same edges, with vertices computed in exactly the same way (without possibility of roundoff errors making the vertices slightly different at different times).

10/21/03 Some examples of scan line interpolation. These usually go away if only triangles are used.

Readings for the above: Buss, Section II.4, pp. 58-66.

Controlling Polygon Rendering in OpenGL (see Project #2, #3)

```
glShadeModel( GL_FLAT );
glShadeModel( GL_SMOOTH );

glCullFace( GL_BACK );
glEnable ( GL_CULL_FACE );
glFrontFace ( GL_CW ):    and   glFrontFace( GL_CCW );

glPolygonMode( GL_FRONT_AND_BACK, GL_LINE );
glPolygonMode( GL_FRONT_AND_BACK, GL_FILL );
glPolygonMode( GL_FRONT_AND_BACK, GL_POINT );
```

Other options instead of GL_FRONT_AND_BACK are available (guess what they are!)

Reading for the above: Buss, Chapter I.

The Phong Lighting Model

Discussion of flat versus smooth shading. Lighting and shading help three-dimensionality of appearance. They hide the facets (flat polygonal faces) that are used to model smooth surfaces.

The teapot with various kinds of lighting applied.

Specular highlights, examples.

Three "kinds" of light:

1. Specular
2. Diffuse
3. Ambient

Four kinds of light reflection from surfaces:

1. Specular - Light reflects mostly in direction of prefect reflection . (That is, glossy, nearly mirror-like).
2. Diffuse - Light reflects in all directions)
3. Ambient - Light has no incoming direction, and reflects equally in all directions.
4. Emissive – Emits light independently of

Red, green and blue components of light and reflection treated independently.
The "Phong lighting" model used by OpenGL is a so-called *local lighting model.* This means that multiple reflections of light from objects, or the shadoing of light by objects are not handled. [Next quarter, in Math 155B, we will study two prominent global lighting models, namely ray tracing and radiosity.]

10/23/03 BRIDF (Bidirectional Reflected Intensity Distribution Function)

Superposition principle.

Intensity of light. Vectors **l, n, v.**

Diffuse Reflection.
Diffuse Reflectivity Coefficient ρ_d
Lambertian object. The moon as example of non-Lambertian.

Specular Reflection.
Specular Reflectivity Coefficient ρ_s
Specular exponent, f,
The Halfway Vector shortcut.
Directional versus positional lights.
Local vs non-local viewer.

Ambient Light
Ambient Reflectivity Coeffient, ρ_a
Emissiveness.

For above material, see Buss III.1.1-1.4 pp.67-75, and Section III.2.1, pp.87-89.

10/30/03

OpenGL commands for specifying lights, materials.

Lots of OpenGL commands.
Enabling Phong lighting.
Assigning properties to lights, including position and direction.
Assigning material properties to surfaces.
Specifying normals.
Distance attenuation.
Spotlight effects.

For the above, see Buss, Section III.1.8, pp. 82-87.

Normal Vectors.

Calculating the normal to a triangle or (planar) quadrangle.
Assigning normals to vertices based on averaging the normals of the adjacent quads or triangles.

Parametric surfaces.
11/4/03 Calculating the normal using cross product of partial derivatives.

Level surface.
Calculating the normal using gradient vectors.

Example of the mushroom cap ellipsoid from Project #3.

How normal vectors transform under linear transformations. (Use the inverse of the transpose of the modelview matrix.)

For the above, see Buss, Section III.1.6-III.1.7, pp. 78-82.

Gouraud Interpolation and Phong Interpolation

Gouraud interpolation shades interiors of lines or triangles or polygons by averaging the colors in RGB space.

Phong interpolation performs shading by interpolating the surface normal at each pixel in the interior, and recomputes the lighting equation at each pixel.

Gouraud interpolation is more commonly used, since it is easier and can be done late in the rendering pipeline, without needing knowledge of the lights and the material properties.

Advantage of Phong shading is that specular highlights in the middle of polygons are not missed or under-emphasized; plus other special effects like spotlights work better.

For the above, see Buss, Section III.1.5, pp. 75-78.

[Cook-Torrance lighting model (section III.2) is skipped in this course.]

11/6/03

Averaging and Linear Interpolation.

Interpolation between two points. LERP-ing

Linear Interpolation between two points.
Lerp(**x, y,** α).
How to find the alpha (α) value for **u** to express as **u** = Lerp(**x, y,** α).
Defining a function by linear interpolation/extrapolation from its value on two points.

Linear combinations, affine combinations, weighted averages.

Definition of linear combinations.
Definition of affine combinations.
Definition of weighted averages.

11/13/03

Thm: Affine combinations are preserved under affine transformations.

Barycentric coordinates

Barycentric coordinate serve as a method of using three values to express a point in a triangle as a weighted sum of the vertices. More generally, any point in the plane containing the triangle can be expressed as an affine combination of the three vertices of the triangle.

Definition of barycentric coordinates.

Basic existence theorem: Any point in the plane containing the triangle has a set of barycentric coordinates that express the point as an affine combination of the vertices. If the point is on on the triangle or in the interior, its barycentric coordinates are non-negative and thus the point is a weight average of the vertices of the triangle.

Area interpretation: Let **u** be a point in the triangle. Then its barycentric coordinates are proportional to the areas of the three triangles form from **u** and two the vertices of the triangle. (Theorem IV.4, page 105).

(This is where the midterm topics end.)

11/20/03 How to calculate barycentric coordinates. Efficient dot product algorithm.

For the above, see Buss, Section IV.1, pp. 99-107..

Bilinear Interpolation.

Basic idea of bilinear interpolation is to warp a unit square into a "patch" based on just the four corners of the patch. The patch may lie in a higher dimensional space and not be flat, but it is "rectangular-ish."
Definition of bilinear interpolation.
Order of lerp-ing does not matter.

(We will skip the material on the projected convexity condition and inverting bilinear interpolation.)

For above material, see Buss, first part of IV.2, pp. 107-109.

Texture Mapping

Basics of texture mapping.
The disadvantages of "source-based" copying, advantages of "destination-based" copying.
Texture coordinates.
Basics of how texture maps are used in OpenGL.
glTexCoord2f(s, t); - specify texture coordinates of next vertex.

Aliasing problems.
Mipmapping.
Supersampling.
Stochastic supersampling.
Jittering.

**gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGBA, width, height,
GL_RGBA, GL_FLOAT, pixelArray);**

gluTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, (or **…_T**)
GL_REPEAT or **GL_CLAMP** or **GL_CLAMP_TO_EDGE);**

**glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST** or **GL_LINEAR);**

**glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_*XXX*_MIPMAP_*YYY***);

where ***XXX*** and ***YYY*** are either "**NEAREST"** or "**LINEAR".**

See the **TextureBMP** project for an example of how to load a texture map from a bitmap (.BMP) file, and the **FourTextures** project for an example of how to handle multiple textures.

11/25/03 Using GL_BLEND, GL_MODULATE rather than GL_DECAL or GL_REPLACE to apply texture maps. Give the underlying surface a white or grey color that modulates the texture map color.

See the **TextureTorus** program for an example of using the modulate/blend option.

Separate specular highlights.

Purpose: put specular highlights on texture mapped surfaces.
Done by keeping specular color, as computed by the Phong model, separate from the ambient, diffuse and emissive components. Texture map only affects non-specular components.

This are not supported by the OpenGL on the lab PCs.

Environment mapping

Simulates reflection by applying a texture map.
Surface normals, or more properly, view reflection directions, determine texture coordinates.
Spherical environment mapping.
Box environment mapping.
See OpenGL documentation for how to implement.

Bump mapping

Virtual displacement of surface: the normals to the surface are manipulated, without actually changing the surface. Gives convincing appearance of "bumps".

Not available in OpenGL (at least current versions).

See textbook by Chapter V, for all the above material on texture mapping.

12/2/03

Color

Trichromatic theory of human color perception
Opponent theory of human color perception.

Additive/linear nature of color (as perceived by humans).

Common methods of representing color.
RGB. (4 bit, 16 bit, 32 bit formats)
CMYK
HSL
“Browser safe colors”