Slides(Mostly) Courtesy – E. Angel and D. Shreiner
Recap from Recent Past
The Sierpinski Gasket
attribute vec4 vPosition;
void main()
{
  gl_Position = vPosition;
}
Sierpinski Fragment Shader

...  
// Load shaders and use the resulting shader program
  GLuint program = InitShader( "vshader21.glsl", "fshader21.glsl" );
  glUseProgram( program );
  ..

void
main()
{
  gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
Fragment vs Vertex Shader

per vertex lighting

per fragment lighting
OpenGL and GLSL

• Shader based OpenGL is based less on a state machine model than a data flow model
• Most state variables, attributes and related pre 3.1 OpenGL functions have been deprecated
• Action happens in shaders
• Job is application is to get data to GPU
GLSL

• C-like with
  – Matrix and vector types (2, 3, 4 dimensional)
  – Overloaded operators
  – C++ like constructors

• Similar to Nvidia’s Cg and Microsoft HLSL
• Code sent to shaders as source code
Still Maximal Portability

• Display device independent

• Window system independent

• Operating system independent
A Few More Things
Hardware Rendering Pipeline

1. Host interface
2. Vertex processing
3. Triangle setup
4. Pixel processing
5. Memory interface

- Raw Vertices & Primitives
- Transformed Vertices & Primitives
- Fragments
- Processed Fragments
- Pixels

Display

[Diagram showing the flow of data through the rendering pipeline]
OpenGL Primitives

- GL_POINTS
- GL_LINES
- GL_LINE_STRIP
- GL_LINE_LOOP
- GL_TRIANGLES
- GL_TRIANGLE_STRIP
- GL_TRIANGLE_FAN
Triangles

- Triangles must be
  - Simple: edges cannot cross
  - Convex: All points on line segment between two points in a polygon are also in the polygon
  - Flat: all vertices are in the same plane
- User must create triangles (triangulation)
- OpenGL contains a tessellator
Space ?

point2 vertices[3] = {point2(0.0, 0.0),
                      point2( 0.0, 1.0), point2(1.0, 1.0)};
Transform Spaces

Object Space  ➔  Screen Space
Coordinate Systems

• The units in **points** can be *object, world, model* or *problem coordinates*

• Viewing specifications are also in object coordinates

• Same for lights

• Eventually pixels will be produced in *window coordinates*
Default Camera

- Camera at origin in object space pointing in -z direction
- Default viewing volume
  - box centered at origin with sides of length 2
Orthographic Viewing

Points projected forward along $z$ axis onto plane $z=0$
Viewports

• Use partial window for image: glViewport(x, y, w, h)
• w, h – pixel coordinates
• x, y – lower corner
Writing Shaders
Simple Vertex Shader

in vec4 vPosition;
void main(void)
{
    gl_Position = vPosition;
}
Execution Model

Vertex data
Shader Program

Application Program

GPU

Vertex Shader

Primitive Assembly

glDrawArrays

Vertex
void main(void) {
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}

Simple Fragment Program
Execution Model

Shader Program

Rasterizer → Fragment Shader → Frame Buffer

Fragment → Fragment Color
Data Types

- **C types**: int, float, bool
- **Vectors**:
  - float vec2, vec3, vec4
  - Also int (ivec) and boolean (bvec)
- **Matrices**: mat2, mat3, mat4
  - Stored by columns
  - Standard referencing m[row][column]
- **C++ style constructors**
  - vec3 a = vec3(1.0, 2.0, 3.0)
  - vec2 b = vec2(a)
Pointers

- There are no pointers in GLSL
- C structs which can be copied back from functions
- Matrices and vectors can be passed to and from GLSL functions, e.g. `mat3 func(mat3 a)`
Selection and Swizzling

• Access array elements-by-element using [] or selection (.) operator with
  – x, y, z, w
  – r, g, b, a
  – s, t, p, q
  – a[2], a.b, a.z, a.p are the same

• Swizziling operator to manipulate components
  vec4 a;
  a.yz = vec2(1.0, 2.0);
Example: Vertex Shader

```cpp
const vec4 red = vec4(1.0, 0.0, 0.0, 1.0);
out vec3 color_out;
void main(void)
{
    gl_Position = vPosition;
    color_out = red;
}
```
in vec3 color_out;

void main(void)
{
    gl_FragColor = color_out;
}

// in latest version use form
// out vec4 fragcolor;
// fragcolor = color_out;
Qualifiers

- GLSL has many qualifiers like `const` as C/C++

- Variables can change
  - Once per primitive
  - Once per vertex
  - Once per fragment
  - At any time in the application

- Vertex attributes are interpolated by the rasterizer into fragment attributes
Passing values

• Call by value-return
• Variables are copied in
• Returned values are copied back
• Two possibilities
  – in
  – out
Attribute Qualifier

- Attribute-qualified variables can change at most once per vertex
- User defined (in application program)
  - Use in qualifier to get to shader
  - in float temperature
  - in vec3 velocity
Uniform Qualified

• Variables that are constant for an entire primitive

• Can be changed in application and sent to shaders

• Cannot be changed in shader

• Used to pass information to shader such as the bounding box of a primitive
Example

GLint aParam;
aParam = glGetUniformLocation(myProgObj, "angle");
/* angle defined in shader */

/* my_angle set in application */
GLfloat my_angle;
my_angle = 5.0 /* or some other value */

glUniform1f(aParam, my_angle);
Varying Qualified

- Variables passed from vertex to fragment shader
  - Automatically interpolated by the rasterizer

- Old style - varying vec4 color
  - Use out in vertex shader and in in fragment shader
    out vec4 color;
Wave Motion Vertex Shader

```glsl
in vec4 vPosition;
uniform float xs, zs; // frequencies
uniform float h; // height scale
void main()
{
  vec4 t = vPosition;
  t.y = vPosition.y + h*sin(time + xs*vPosition.x) + h*sin(time + zs*vPosition.z);
  gl_Position = t;
}
```
Particle System

```cpp
in vec3 vPosition;
uniform mat4 ModelViewProjectionMatrix;
uniform vec3 init_vel;
uniform float g, m, t;
void main()
{
    vec3 object_pos;
    object_pos.x = vPosition.x + vel.x*t;
    object_pos.y = vPosition.y + vel.y*t
                   + g/(2.0*m)*t*t;
    object_pos.z = vPosition.z + vel.z*t;
    gl_Position =
                  ModelViewProjectionMatrix*vec4(object_pos, 1);
}
```
Fragment Shader

/* pass-through fragment shader */

in vec4 color;
void main(void)
{
    gl_FragColor = color;
}

Vertex Shader Applications

• Moving vertices
  – Morphing
  – Wave motion
  – Fractals

• Lighting
  – More realistic models
  – Cartoon shaders
Operators and Functions

• Standard C functions
  – Trigonometric
  – Arithmetic
  – Normalize, reflect, length

• Overloading of vector and matrix types
  mat4 a;
  vec4 b, c, d;
  c = b*a; // a column vector stored as a 1d array
  d = a*b; // a row vector stored as a 1d array
Adding Color

• Send color to the shaders as a vertex attribute or as a uniform variable
• Choice depends on frequency of change
• Associate a color with each vertex
• Set up an array of same size as positions
• Send to GPU as a vertex buffer object
typedef vec3 color3;
color3 base_colors[4] = {color3(1.0, 0.0, 0.0), ....
color3 colors[NumVertices];
vec3 points[NumVertices];

//in loop setting positions

colors[i] = basecolors[color_index]
position[i] = ........
Setting Up Buffer Object

//need larger buffer

glBufferData(GL_ARRAY_BUFFER, sizeof(points) + sizeof(colors), NULL, GL_STATIC_DRAW);

//load data separately

glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(points), points);
glBufferSubData(GL_ARRAY_BUFFER, sizeof(points), sizeof(colors), colors);
Second Vertex Array

// vPosition and vColor identifiers in vertex shader

loc = glGetAttribLocation(program, "vPosition");
glEnableVertexAttribArray(loc);
glVertexAttribPointer(loc, 3, GL_FLOAT, GL_FALSE, 0,
BUFFER_OFFSET(0));

loc2 = glGetAttribLocation(program, "vColor");
glEnableVertexAttribArray(loc2);
glVertexAttribPointer(loc2, 3, GL_FLOAT, GL_FALSE, 0,
BUFFER_OFFSET(sizeof(points)));
Next Topic – Linear Algebra
Vectors

• Physical definition:
  – Direction
  – Magnitude

• Examples
  – Light Direction
  – View Direction
  – Normal
Abstract Spaces

- Scalars
- (Linear) Vector Space
  - Scalars and vectors
- Affine Space
  - Scalars, vectors, and points
- Euclidean Space
  - Scalars, vectors, points
  - Concept of distance
- Projections
Vectors – Linear Space

• Every vector
  – has an inverse
  – can be multiplied by a scalar
• There exists a zero vector
  – Zero magnitude, undefined orientation
• The sum of any two vectors is a vector - closure
Vector Spaces

- **Vectors** = \( n \)-tuples
- **Vector-vector addition**
- **Scalar-vector multiplication**
- **Vector space:**
Linear Independence

\[ \alpha_1 \vec{u}_1 + \alpha_2 \vec{u}_2 + \cdots + \alpha_n \vec{u}_n = 0 \quad \text{iff} \]

\[ \alpha_1 = \alpha_2 = \cdots = \alpha_n = 0 \]

\[ p = (x, y, z) = x\vec{i} + y\vec{j} + z\vec{k} \]
Vector Spaces

• **Dimension**
  - The greatest number of linearly independent vectors

• **Basis** \( \{ \beta_i \} \)
  - \( n \) linearly independent vectors (\( n \): dimension)

• **Representation** \( \vec{v} = \beta_1 \vec{v}_1 + \beta_2 \vec{v}_2 + \cdots + \beta_n \vec{v}_n \)
  - Unique expression in terms of the basis vectors

• **Change of Basis: Matrix \( M \)**
  - Other basis \( \vec{v}_1', \vec{v}_2', \ldots, \vec{v}_n' \)
  \[ \vec{v} = \beta_1' \vec{v}_1' + \beta_2' \vec{v}_2' + \cdots + \beta_n' \vec{v}_n' \]
Vectors

• These vectors are identical
  – Same length and magnitude

• Vectors spaces insufficient for geometry
  – Need points
Points

• Location in space
• Operations allowed between points and vectors
  – Point-point subtraction yields a vector
  – Equivalent to point-vector addition

\[ \vec{v} = P - Q \]
\[ P = \vec{v} + Q \]

\[(P - Q) + (Q - R) = (P - R)\]
**Affine Spaces**

*Frame*: a **Point** $P_0$ and a Set of **Vectors** $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n$

Representations of the vector and point: $n$ scalars

<table>
<thead>
<tr>
<th>Vector</th>
<th>$\mathbf{v} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \cdots + \alpha_n \mathbf{v}_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>$\mathbf{P} = P_0 + \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2 + \cdots + \beta_n \mathbf{v}_n$</td>
</tr>
</tbody>
</table>
Affine Spaces

- Point + a vector space
- Operations
  - Vector-vector addition
  - Scalar-vector multiplication
  - Point-vector addition
  - Scalar-scalar operations
- For any point define
  - 1 \cdot P = P
  - 0 \cdot P = \mathbf{0} (zero vector)
Question

How Far Apart Are Two Points in Affine Spaces?

Operation: Inner (dot) Product
Euclidean (Metric) Spaces

- **Magnitude (length)** of a vector
  \[ |v| = \sqrt{v \cdot v} \]

- **Distance** between two points
  \[ |P - Q| = \sqrt{(P - Q) \cdot (P - Q)} \]

- Measure of the angle between two vectors
  \[ u \cdot v = |u||v|\cos \theta \]

- $\cos \theta = 0 \rightarrow$ orthogonal
- $\cos \theta = 1 \rightarrow$ parallel
In Pictures

\[
\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \cdot \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = a_x b_x + a_y b_y + a_z b_z
\]

Definition

\[
\begin{align*}
\mathbf{a} &= (a_x, a_y, a_z) \\
\mathbf{b} &= (b_x, b_y, b_z) \\
\mathbf{a} \cdot \mathbf{b} &= a_x b_x + a_y b_y + a_z b_z \\
\end{align*}
\]

Dot Product (Inner Product) (Scalar Product)

Geometrical Interpretation

\[
\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta
\]

relationship with angle

\[
0 < \mathbf{a} \cdot \mathbf{b}
\]

\[
\mathbf{a} \cdot \mathbf{b} = 0
\]

\[
\mathbf{a} \cdot \mathbf{b} < 0
\]
Euclidean Spaces

- Combine two vectors to form a real
- $\alpha$, $\beta$, $\gamma$, ...: scalars, $u$, $v$, $w$, ...: vectors

$$u \cdot v = v \cdot u$$

$$(\alpha u + \beta v) \cdot w = \alpha u \cdot w + \beta v \cdot w$$

$v \cdot v > 0 \text{ if } v \neq 0$

$0 \cdot 0 = 0$

**Orthogonal:** $u \cdot v = 0$
Projections

- Problem: Find shortest distance from a point to a line on a plane

- Given Two Vectors $w = \alpha v + u$
  - Divide into two parts: one parallel and one orthogonal

\[
\begin{align*}
  w \cdot v &= \alpha v \cdot v + u \cdot v = \alpha v \cdot v \\
  \therefore \alpha &= \frac{w \cdot v}{v \cdot v} \\
  \therefore u &= w - \alpha v = w - \frac{w \cdot v}{v \cdot v} v
\end{align*}
\]

Projection of one vector onto another
Making New Vectors
Cross Product

\[ c = a \times b = \begin{vmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = (a_2b_3 - a_3b_2)i + (a_3b_1 - a_1b_3)j + (a_1b_2 - a_2b_1)k \]

The direction of \( c \) can also be obtained from the right hand rule.
Cross Product

Definition
\[ a = (a_1, a_2, a_3) \quad b = (b_1, b_2, b_3) \]

Cross Product
(Outer Product)
(Vector Product)
\[ a \times b = \begin{pmatrix}
  a_2 b_3 - a_3 b_2 \\
  a_3 b_1 - a_1 b_3 \\
  a_1 b_2 - a_2 b_1
\end{pmatrix} \]

Geometrical Interpretation
- Perpendicular Direction

- Length & Area
\[ |a \times b| = |a||b| \sin \theta \]
Area of Parallelogram
\[ = |a||b| \sin \theta \]
Area of Triangle
\[ = \frac{1}{2} |a||b| \sin \theta \]
- Parallel
\[ a \parallel b \quad |a \times b| = 0 \]
\[ A = 180^\circ \quad |a \times b| = 0 \]
Parametric Forms
Lines, Rays

- Consider all points of the form
  - $P(\alpha) = P_0 + \alpha \mathbf{d}$
  - Set of all points that pass through $P_0$ in the direction of the vector $\mathbf{d}$
2D Forms for lines

- Two-dimensional forms
  - Explicit: $y = mx + h$
  - Implicit: $ax + by + c = 0$
  - Parametric:
    \[
    x(a) = ax_0 + (1-a)x_1 \\
y(a) = ay_0 + (1-a)y_1
    \]
Rays, Line Segments

If $a \geq 0$, then $P(a)$ is the ray leaving $P_0$ in the direction $d$

If we use two points to define $v$, then

$P(a) = Q + a(R-Q) = Q + av$

$= aR + (1-a)Q$

For $0 \leq a \leq 1$ we get all the points on the line segment joining $R$ and $Q$
Curves

\[
\begin{align*}
\begin{cases}
x(t) &= r \frac{1 - t^2}{1 + t^2} \\
y(t) &= r \frac{2t}{1 + t^2}
\end{cases}
\quad 0 \leq t \leq 2\pi,
\end{align*}
\]

- circle: \( x^2 + y^2 - r^2 = 0 \)
- ellipse: \( \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \)
- hyperbola: \( \frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \)
- parabola: \( y^2 - 2px = 0 \)
Planes

Defined by a point and two vectors or by three points

\[ P(a,b) = R + au + bv \]

\[ P(a,b) = R + a(Q-R) + b(P-Q) \]
Triangles

For \(0 \leq \alpha, \beta \leq 1\), we get all points in triangle \(T(\alpha, \beta)\).

Convex sum of \(P\) and \(Q\)

Convex sum of \(S(\alpha)\) and \(R\)
Barycentric Coordinates

(0, 0, 1)

(0, 1/2, 1/2)

(1/2, 1/2, 1/2)

(1/4, 1/4, 1/2)

(1/2, 1/4, 1/2)

(1/2, 1/2, 0)

(1, 0, 0)
Barycentric Coordinates

Triangle is convex
Any point inside can be represented as an affine sum

\[ P(\alpha_1, \alpha_2, \alpha_3) = \alpha A + \beta B + \gamma C \]

where

\[ \alpha + \beta + \gamma = 1 \]
\[ \alpha, \beta, \gamma \geq 0 \]
Barycentric Coordinates

Calculating Areas?

\[ \alpha = \frac{A_{PBC}}{A_{ABC}} \]
\[ \beta = \frac{A_{PCA}}{A_{ABC}} \]
\[ \gamma = 1 - \alpha - \beta \]
Matrices
Matrices

- Definitions
- Matrix Operations
- Row and Column Matrices
- Rank
- Change of Representation
- Cross Product
What is a Matrix?

Elements, organized into rows and columns

\[
\begin{bmatrix}
a & b \\
c & d
\end{bmatrix}
\]
Definitions

\[ A = \begin{bmatrix} a_{ij} \end{bmatrix} \]

\( n \times m \) Array of Scalars (\( n \) Rows and \( m \) Columns)
- \( n \): row dimension of a matrix, \( m \): column dimension
- \( m = n \): square matrix of dimension \( n \)
- Element
  \( \{a_{ij}\}, \; i = 1, \ldots, n, \; j = 1, \ldots, m \)
- Transpose: interchanging the rows and columns of a matrix
  \[ A^T = \begin{bmatrix} a_{ji} \end{bmatrix} \]

- Column Matrices and Row Matrices
  - Column matrix (\( n \times 1 \) matrix):
  \[ \mathbf{b} = \begin{bmatrix} b_i \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \]
  - Row matrix (\( 1 \times n \) matrix):
    \[ \mathbf{b}^T \]
Basic Operations

Addition, Subtraction, Multiplication

\[
\begin{pmatrix}
a & b \\
c & d \\
\end{pmatrix}
+ \begin{pmatrix}
e & f \\
g & h \\
\end{pmatrix} = \begin{pmatrix}
a+e & b+f \\
c+g & d+h \\
\end{pmatrix}
\]

add elements

\[
\begin{pmatrix}
a & b \\
c & d \\
\end{pmatrix}
- \begin{pmatrix}
e & f \\
g & h \\
\end{pmatrix} = \begin{pmatrix}
a-e & b-f \\
c-g & d-h \\
\end{pmatrix}
\]

subtract elements

\[
\begin{pmatrix}
a & b \\
c & d \\
\end{pmatrix}
\begin{pmatrix}
e & f \\
g & h \\
\end{pmatrix} = \begin{pmatrix}
ae+bg & af+bh \\
ce+dg & cf+dh \\
\end{pmatrix}
\]

Multiply each row by each column
Matrix Operations

+ **Scalar-Matrix Multiplication**
\[
\alpha A = [\alpha a_{ij}]
\]

+ **Matrix-Matrix Addition**
\[
C = A + B = [a_{ij} + b_{ij}]
\]

+ **Matrix-Matrix Multiplication**

  + A: \(n \times l\) matrix, B: \(l \times m\) ➞ C: \(n \times m\) matrix
  \[
  C = AB = [c_{ij}]
  \]
  \[
  c_{ij} = \sum_{k=1}^{l} a_{ik} b_{kj}
  \]
Matrix Operations

+ Properties of Scalar-Matrix Multiplication
  \[ \alpha(\beta A) = (\alpha \beta)A \]
  \[ \alpha \beta A = \beta \alpha A \]

+ Properties of Matrix-Matrix Addition
  + Commutative: \[ A + B = B + A \]
  + Associative: \[ A + (B + C) = (A + B) + C \]

+ Properties of Matrix-Matrix Multiplication
  \[ A(BC) = (AB)C \]
  \[ AB \neq BA \]

+ **Identity Matrix** \( I \) (Square Matrix)
  \[ I = \begin{bmatrix} a_{ij} \end{bmatrix}, \quad a_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \]
  \[ AI = A \]
  \[ IB = B \]
Identity Matrix

\[ I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]
Multiplication

• Is $AB = BA$? Maybe, but maybe not!

\[
\begin{bmatrix}
a & b \\
c & d
\end{bmatrix}\begin{bmatrix}
e & f \\
g & h
\end{bmatrix} = \begin{bmatrix} ae + bg & \ldots \\ \ldots & \ldots \end{bmatrix} \quad \begin{bmatrix}
e & f \\
g & h
\end{bmatrix}\begin{bmatrix}
a & b \\
c & d
\end{bmatrix} = \begin{bmatrix} ea + fc & \ldots \\ \ldots & \ldots \end{bmatrix}
\]

• Heads up: multiplication is NOT commutative!
Row and Column Matrices

- Column Matrix
  - $p^T$: row matrix

- Concatenations
  - Associative

- By Row Matrix

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}
\]

\[
p = \begin{bmatrix}
  z \\
  y \\
  x
\end{bmatrix}
\]

\[
p' = Ap
\]

\[
p' = ABCp
\]

\[
(AB)^T = B^T A^T
\]

\[
p'^T = p^T C^T B^T A^T
\]
Inverse of a Matrix

- Identity matrix:
  \[ AI = A \]

- Some matrices have an inverse, such that:
  \[ AA^{-1} = I \]

- Inversion is tricky:
  \[ (ABC)^{-1} = C^{-1}B^{-1}A^{-1} \]

- Derived from non-commutativity property
Determinant of a Matrix

• Used for inversion

• If det(A) = 0, then A has no inverse

• Can be found using factorials, pivots, and cofactors!

• And for Areas of Triangles

\[ A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \]

\[ \text{det}(A) = ad - bc \]

\[ A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \]
Area of Triangle – Cramer’s Rule

\[ \text{Ans} = \frac{1}{2} \cdot \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \]
Use This Here

\[ \alpha = \frac{A_{pCB}}{A_\circ} \]

\[ \beta = \frac{A_{pCA}}{A_\circ} \]

\[ \gamma = 1 - \alpha - \beta \]
Transformations