Today's Outline

Review of basic geometry
- Points and vectors
- Dot product, cross product
- Coordinate systems and frames
- libst STVector{2,3} and STPoint{2,3}

Drawing curves
- Parametric curves
- Implicit curves

Triangle rasterization
Quick Review of Geometry

Vectors and the Parallelogram Rule

Vector addition define for any number of dimensions
Dot Product

\[ a \cdot b = |a||b| \cos \phi \]

The projection of \( a \) onto \( b \)

N. B. the projection is 0 if \( a \) is perpendicular to \( b \)

Orthonormal Vectors

Perpendicular \[ x \cdot y = 0 \]

Unit length \[ x \cdot x = 1 \]
\[ y \cdot y = 1 \]
Coordinates wrt Basis Vectors

\[ c = \alpha x + \beta y \]

\[ \alpha = x \cdot c = \alpha x \cdot x + \beta x \cdot y \]
\[ \beta = y \cdot c = \alpha y \cdot x + \beta y \cdot y \]

Dot Product Between Two Vectors

\[ a = x_a x + y_a y \]
\[ b = x_b x + y_b y \]
\[ a \cdot b = x_a x_b + y_a y_b \]
\[ a \cdot a = x_a^2 + y_a^2 = |a|^2 \]
\[ |a| = \sqrt{x_a^2 + y_a^2} = \sqrt{a \cdot a} \]
Reference Frame

An origin $o$ and two unit vectors $x$, $y$ define a frame of reference

Scalar Cross Product

$|a \times b| = |a||b| \sin \phi$

Equal to the area of the parallelogram (base $x$ height) defined by $a$ and $b$
Vector Cross Product

\[ \mathbf{c} = \mathbf{a} \times \mathbf{b} \]

\[ x_c = y_a z_b - z_a y_b \]
\[ y_c = z_a x_b - x_a z_b \]
\[ z_c = x_a y_b - z_a x_b \]

\( \mathbf{c} \) perpendicular to both \( \mathbf{a} \) and \( \mathbf{b} \)

\( |\mathbf{c}| \) is equal to the area of quadrilateral \( \mathbf{a} \ \mathbf{b} \)

---

Vector Cross Product

\[ \mathbf{x} \times \mathbf{y} = \mathbf{z} \]
\[ \mathbf{y} \times \mathbf{z} = \mathbf{x} \]
\[ \mathbf{z} \times \mathbf{x} = \mathbf{y} \]
\[ \mathbf{x} \times \mathbf{x} = 0 \]
\[ \mathbf{y} \times \mathbf{y} = 0 \]
\[ \mathbf{z} \times \mathbf{z} = 0 \]

Right-Hand Rule
3D Not that Different than 2D

typedef float float2[2];
typedef float float3[3];

float2 p2;
float3 p3;

glVertex2fv( p2 );
glVertex3fv( p3 );

Vector Operations

Vectors: \( \mathbf{u}, \mathbf{v}, \mathbf{w} \)

\[ \langle \text{Vector} \rangle = \langle \text{Scalar} \rangle \times \langle \text{Vector} \rangle \]

\[ \mathbf{v} = \alpha \mathbf{w} \]

\[ \langle \text{Vector} \rangle = \langle \text{Vector} \rangle + \langle \text{Vector} \rangle \]

\[ \mathbf{u} = \mathbf{v} + \mathbf{w} \]

Implementation of parallelogram rule
Point Operations

Points: \( p, q, r \)

\[
\langle \text{Point} \rangle = \langle \text{Point} \rangle + \langle \text{Vector} \rangle
\]
\[
q = p + v
\]

\[
\langle \text{Vector} \rangle = \langle \text{Point} \rangle - \langle \text{Point} \rangle
\]
\[
v = q - p
\]

A point is an origin and a vector displacement

Illegal Operations

\[
\langle \text{Point} \rangle = \langle \text{Scalar} \rangle \times \langle \text{Point} \rangle
\]
\[
p = \alpha q
\]

\[
\langle \text{Point} \rangle = \langle \text{Point} \rangle + \langle \text{Point} \rangle
\]
\[
p = q + r
\]

\[
\langle \text{Vector} \rangle = \langle \text{Point} \rangle + \langle \text{Vector} \rangle
\]
\[
v = p + w
\]

\[
\langle \text{Point} \rangle = \langle \text{Point} \rangle - \langle \text{Point} \rangle
\]
\[
p = q - r
\]
STVector and STPoint Types

C++ classes

class STVector2;
class STPoint2;

class STVector3;
class STPoint3;

Overloaded operators for point and vector ops
Methods on points and vectors are different
See vect.cpp
Parametric Curves

Position determined by scalar parameter $t$

$$p(t) = x(t)x + y(t)y$$

Example: circle

$$p(\theta) = \cos \theta x + \sin \theta y$$

Rolling Circles = Epicycloid

$$a=3$$
$$b=1$$

Epicycloid

\[ c = ((a + b) \cos t, (a + b) \sin t) \]
Epicycloid

\[ p = c + (b \cos(s + t), b \sin(s + t)) \]

at = bs

Demo of Epicycloid
Implicit Curves

Position classified

\[ f(x, y) \begin{cases} < 0 & \text{inside} \\ = 0 & \text{on} \\ > 0 & \text{outside} \end{cases} \]

Example: circle

\[ f(x, y) = x^2 + y^2 - 1 \]

Directed Line

\[ t = p_1 - p_0 = (x_1 - x_0, y_1 - y_0) \]
Perpendicular Vector in 2D

\[ \text{Perp}((x,y)) = (-y,x) \]

Line Equation

\[ (p - p_0) \cdot n = 0 \]

This equation must be true for all point \( p \) on the line.
Normal to the Line

\[ t = p_1 - p_0 = (x_1 - x_0, y_1 - y_0) \]
\[ n = \text{Perp}(t) = (y_0 - y_1, x_1 - x_0) \]

Line Equation

\[ n = (A, B) \]
\[ A x + B y + C = 0 \]
\[ A = y_1 - y_0 \]
\[ B = x_0 - x_1 \]
\[ C = x_0 y_1 - y_0 x_1 \]
Line Divides Plane into 2 Half-Spaces

Normal n points to the left of the line
Inside (negative values) to the right

Triangle Rasterization
Triangle Rasterization

Output fragment if pixel center is inside the triangle

```
rasterize( vert v[3] )
{
    bbox b; bound3(v,b);
    for( int y=b.ymin; y<b.ymax, y++ )
        for( int x=b.xmin; x<b.xmax, x++ )
            if( inside3(v,x,y) )
                fragment(x,y);
}

GPUs contain triangle rasterization hardware
Can output billions of fragments per second
```
Compute Bounding Rectangle (Box)

```
bounds3( vert v[3], bbox& b )
{
    b.xmin = ceil(min(v[0].x, v[1].x, v[2].x));
    b.xmax = ceil(max(v[0].x, v[1].x, v[2].x));
    b.ymin = ceil(min(v[0].y, v[1].y, v[2].y));
    b.ymax = ceil(max(v[0].y, v[1].y, v[2].y));
}
```

Calculate tight bound around the triangle
Round coordinates upward (ceil) to the nearest integer

Point Inside Triangle Test

```
rasterize( vert v[3] )
{
    bbox b; bounds3(v, b);
    line 10, 11, 12;
    makeline(&v[0], &v[1], &12);
    makeline(&v[1], &v[2], &10);
    makeline(&v[2], &v[0], &11);
    for (y = b.ymin; y < b.ymax, y++) {
        for (x = b.xmin; x < b.xmax, x++) {
            e0 = 10.0 * x + 10.0 * y + 10.0;
            e1 = 11.0 * x + 11.0 * y + 11.0;
            e2 = 12.0 * x + 12.0 * y + 12.0;
            if (e0 <= 0 && e1 <= 0 && e2 <= 0)
                fragment(x, y);
        }
    }
}
```
Line Equation

Inside on the left for CCW polygons

\[
\text{makeline( vert& v0, vert& v1, line& l )}
\{
    l.a = v1.y - v0.y;
    l.b = v0.x - v1.x;
    l.c = -(l.a * v0.x + l.b * v0.y);
\}
\]

Singularities

Singularities: Edges that touch pixels (e == 0)
Causes two fragments to be generated
  ■ Wasted effort
  ■ Problems with transparency (later lecture)

Not including singularities (e < 0) causes gaps
Handling Singularities

Create shadowed edges (thick lines)
Don’t draw pixels on shadowed edges
Solid drawn; hollow not drawn

```
int shadow( value a, value b ) {
    return (a>0) || (a==0 && b > 0);
}
int inside( value e, value a, value b ) {
    return (e == 0) ? !shadow(a,b) : (e < 0);
}
```

Triangle Rasterization

```
rasterize( vert v[3] )
{
    bbox b;

    bound3(v,b);
    for( int y=b.ymin; y<b.ymax, y++ )
        for( int x=b.xmin; x<b.xmax, x++ )
            if( inside3(v,x,y) )
                fragment(x,y);
}
```

Tested points indicated by filled circles
Don’t need to test hollow circles
Things to Remember

Basic geometry
- Points and vectors
- Operations between points and vectors
- Coordinate frames

Drawing curves
- Parametric curves
  - Break into line segments
  - More about curves latter; Bezier curves
- Implicit curves
  - Drawing pixels on the curve, f(x,y) = 0, is hard
  - Triangle rasterization; draw pixels inside