Today’s Outline

Review of basic geometry
- Points and vectors
- Dot product, cross product
- libst class STVector{23} and STPoint{23}

Drawing curves
- Parametric curves
- Implicit curves

Triangle rasterization
class STShape
Review of Geometry

Vectors and the Parallelogram Rule

Works in any number of dimensions; e.g. 2 or 3
Dot Product

The projection of \( \mathbf{a} \) onto \( \mathbf{b} \)

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

Orthonormal Vectors

\[
\begin{align*}
|\mathbf{x}| &= 1 \\
|\mathbf{y}| &= 1 \\
\mathbf{x} \cdot \mathbf{y} &= 0 \\
\mathbf{x} \cdot \mathbf{x} &= 1 \\
\mathbf{y} \cdot \mathbf{y} &= 1
\end{align*}
\]
Coordinates

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

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

Dot Product Between Two Vectors

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

\[
\begin{align*}
  x \times y &= z \\
  y \times z &= x \\
  z \times x &= y \\
  x \times x &= 0 \\
  y \times y &= 0 \\
  z \times z &= 0
\end{align*}
\]

Right-Hand Rule
Vector Cross Product

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

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

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

|\( \vec{c} \)\| is equal to the area of quad \( \vec{a} \ \vec{b} \)

3D Not that Different than 2D

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

float2 p2;
float3 p3;

glVertex2fv( p2 );
glVertex3fv( p3 );
```
Vector Operations

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

\[
\begin{align*}
\langle \text{Vector} \rangle &= \langle \text{Scalar} \rangle \times \langle \text{Vector} \rangle \\
\mathbf{v} &= \alpha \mathbf{w} \\
\langle \text{Vector} \rangle &= \langle \text{Vector} \rangle + \langle \text{Vector} \rangle \\
\mathbf{u} &= \mathbf{v} + \mathbf{w}
\end{align*}
\]

Points and Vectors are Different!

Vectors have no origin (may be shifted without effect)
Points are formed by picking an origin and adding a vector

Next lecture:
Points and vectors are transformed differently
Point Operations

Points: \( p, q, r \)

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

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

Points and Vectors have Different Ops.

Illegal operations

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

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

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

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

C++ classes

class STVector2;
class STPoint2;

class STVector3;
class STPoint3;

Overload operators to implement vector arithmetic
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

![Circle Parametric Equation](http://upload.wikimedia.org/wikipedia/en/a/a5/EpitrochoidOn3.gif)

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

Rolling Circles = Epicycloid

![Rolling Circles Image](http://upload.wikimedia.org/wikipedia/en/a/a5/EpitrochoidOn3.gif)

$a=3$
$b=1$
Epicycloid

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

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

Demo of epicycloid.c
Implicit Curves

Position classified

\[ f(x, y) \]

\[ \begin{aligned} &< 0 \quad \text{inside} \\ &= 0 \quad \text{on} \\ &> 0 \quad \text{outside} \end{aligned} \]

Example: circle

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

Directed Line

\[ p_0 = (x_0, y_0) \]

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

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

Normal to the Line

Let \( p_0 = (x_0, y_0) \) and \( p_1 = (x_1, y_1) \) be two points on the line. Then the vector \( t = p_1 - p_0 = (x_1 - x_0, y_1 - y_0) \) is parallel to the line. The normal vector \( n \) to the line is given by:

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

The equation must be true for any point $p$:

$$(p - p_0) \cdot n = 0$$

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

Line Divides Plane into 2 Half-Spaces
Line Divides Plane into 2 Half-Spaces

CCW polygon convention: inside on the left

Line Equation

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

Flip to get correct sign
Line Equation

\[ n=(A,B) \]

\[ Ax + By + C = 0 \]

\[ (x_0, y_0) \]

\[ n = (A, B) \]

\[ A = y_1 - y_0 \]
\[ B = x_0 - x_1 \]
\[ C = x_0 y_1 - y_0 x_1 \]
Triangle Rasterization

This is done in hardware on a modern GPU
Can output billions of fragments per second
Compute Bounding Rectangle (Box)

```
bound3( 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;

    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);
}
```

Test points indicated by filled circles
Don’t need to test hollow circles (x<b.xmax, y<b.ymax)
Point Inside Triangle Test

```
rasterize( vert v[3] )
{
    bbox b; bound3(v, b);
    line l0, l1, l2;
    makeline(&v[0],&v[1],&l2);
    makeline(&v[1],&v[2],&l0);
    makeline(&v[2],&v[0],&l1);
    for( y=b.ymin; y<b.ymax, y++ ) {
        for( x=b.xmin; x<b.xmax, x++ ) {
            e0 = l0.A * x + l0.B * y + l0.C;
            e2 = l2.A * x + l2.B * y + l2.C;
            if( e0<=0 && e1<=0 && e2<=0 )
                fragment(x,y);
        }
    }
}
```

Singularities

Including edges cause double hits (e <= 0)
- Wasted effort
- Transparency
- ...

Not including edges (e < 0) causes gaps
- Missing pixels
Handling Singularities

Illuminate polygons from the left-below
Shadowed edges are drawn as thick lines
Don’t draw pixels on shadowed edges (hollow circles)

```
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);
}
```

STShape
Points/Polygons

typedef float Point[3];

Point verts[8] = {
{ -1., -1., -1. },
{  1., -1., -1. },
{  1.,  1., -1. },
{ -1.,  1., -1. },
{ -1., -1.,  1. },
{  1., -1.,  1. },
{  1.,  1.,  1. },
{ -1.,  1.,  1. }
};

int polys[6][4] = {
{ 0, 3, 2, 1 },
{ 2, 3, 7, 6 },
{ 0, 4, 7, 3 },
{ 1, 2, 6, 5 },
{ 4, 5, 6, 7 },
{ 0, 1, 5, 4 }
};

face(int poly[4]) {
    glBegin(GL_POLYGON);
    glVertex3fv(poly[0]);
    glVertex3fv(poly[1]);
    glVertex3fv(poly[2]);
    glVertex3fv(poly[3]);
    glEnd();
}

cube() {
    for( int i = 0; i < n; i++ )
        face(polys[i]);
}

Demo of Shape.c
Things to Remember

Basic geometry
- Points and vectors
- Operations between points and vectors
- Geometric calculations

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

libst (STVector, STPoint, STShape, …)