Vertex Shader Example 2

```
<!DOCTYPE html>
<html>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec3 vNormal;

uniform mat4 modelViewMatrix;
uniform mat4 normalMatrix;
uniform mat4 projectionMatrix;

uniform vec4 lightPosition;
uniform vec4 lightColor;
uniform vec4 materialColor;
uniform float Ka, Kd, Ks, shininess;

varying vec4 fColor;
```

Vertex Shader Example 2

```
void main()
{
    gl_Position = projectionMatrix * modelViewMatrix * vPosition;

   // Transform vertex normal into eye coordinates
    vec3 pos = (modelViewMatrix * vPosition).xyz;
    vec3 N = normalize( (normalMatrix*vec4(vNormal,0.0)).xyz);

    // get light direction and eye direction
    vec3 L = vec3(normalize(lightPosition.xyz-pos)) ;
    vec3 E = normalize(-pos.xyz) ;

    // compute reflected light direction
    vec3 R = reflect(-L, N) ;

    // Compute terms in the illumination equation
    vec4 ambientComponent = Ka*materialColor;
    vec4 diffuseComponent = Kd*max( dot(L, N), 0.0 )*materialColor;
    vec4 specularComponent = vec4(0.0, 0.0, 0.0, 1.0);
    if( dot(L, N) > 0.0 ) {
    float Ks = pow( max(dot(R, E), 0.0), shininess );
        specularComponent = Ks * vec4(1.0, 1.0, 1.0, 1.0);
    }
    fColor = (ambientComponent + diffuseComponent + specularComponent)*lightColor;
    fColor.a = 1.0;
}
</script>
```

Fragment Shader Example 2

```
<script id="fragment-shader" type="x-shader/x-fragment">

precision mediump float;

varying vec4 fColor;

void
main()
{

    gl_FragColor = fColor;

}
</script>
```

Vertex Shader Example 3

```
attribute vec4 vPosition;
attribute vec3 vNormal;

uniform mat4 modelViewMatrix;
uniform mat4 normalMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;

varying vec3 fN, fL, fE;

void main()
{
    // Transform vertex and normal into eye coordinates
    vec3 pos = (modelViewMatrix * vPosition).xyz;
    fN = normalize((normalMatrix*vec4(vNormal,0.0)).xyz);

    // get light direction and eye direction
    fL = vec3(normalize(lightPosition.xyz-pos));
    fE = normalize(-pos.xyz);

    gl_Position = projectionMatrix * modelViewMatrix * vPosition;
}
```

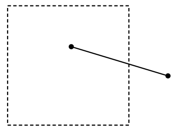
Fragment Shader Example 3

```
precision mediump float;

uniform vec4 lightColor;
uniform vec4 materialColor;
uniform float Ka, Kd, Ks, shininess;

varying vec3 fN, fL, fE;
```

Fragment Shader Example 3

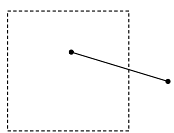
```
void main()
{
    vec3 N = normalize(fN);
    vec3 L = normalize(fL);
    vec3 E = normalize(fE);

    // compute reflected light direction
    vec3 R = reflect(-L, N) ;

    // Compute terms in the illumination equation
    vec4 ambientComponent = Ka*materialColor;
    vec4 diffuseComponent = Kd*max( dot(L, N), 0.0 )*materialColor;
    vec4 specularComponent = vec4(0.0, 0.0, 0.0, 1.0);
    if( dot(L, N) > 0.0 ) {
        float Ks = pow( max(dot(R, E), 0.0), shininess );
        specularComponent = Ks * vec4(1.0, 1.0, 1.0, 1.0);
    }
    vec4 fColor = (ambientComponent+diffuseComponent+specularComponent)*lightColor;
    fColor.a = 1.0;
    gl_FragColor = fColor;
}
```

Clipping



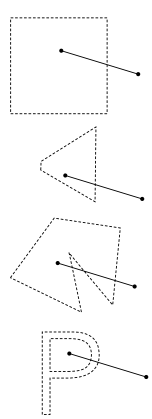
- removing parts of object that are outside field of view
- related terms:
  - culling: quickly determining in/out
  - bounding box: axis-aligned box containing object
  - scissoring: combing clipping with scan conversion

Why Clip?



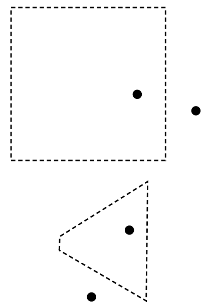
- speed
- hardware cannot handle precision
- correctness (3D perspective)

Clipping Region?



- rectangular, axis-aligned
- convex
- concave (with holes)

Clipping Points

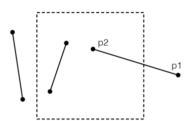


- Rectangular: trivial
- convex: straightforward
  insert point in edge equation
  (must be inside all edges)

  2D: a·x+b·y+c = ?

  3D: a·x+b·y+c·z +d = ?

Clipping Lines: Cohen-Sutherland



1. compute labels for p1 & p2
2. determine if total visible or trivial reject
3. if p1 not outside, swap p1 & p2
4. find edge p1 is out
5. replace p1 with intersection of p1-p2 and edge
6. compute new label for p1

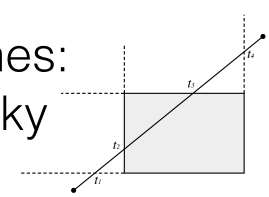
1001	1000	1100
0001	0000	0100
0011	0010	0110

if both labels 0
→trivial accept

if label(p1) ∩ label(p2) ≠ 0
→trivial reject

Hardware acceleration

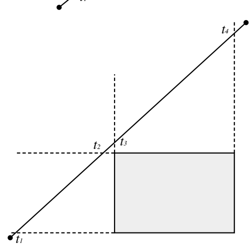
Clipping Lines: Liang-Barsky



Use parametric line equation

$$p_t = p_1 + t\cdot(p_2 - p_1) = p_1 + t\cdot\vec{d}$$

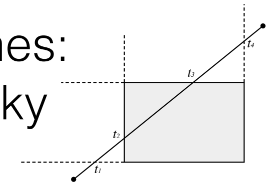
Compute t values for each edge intersection



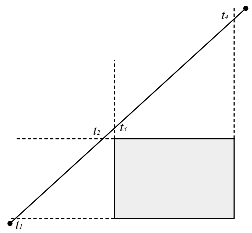
Can classify t values as entering/exiting window edge

To be visible order must be: enter, enter, exit, exit

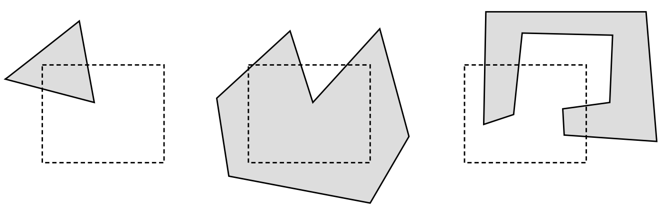
Clipping Lines: Liang-Barsky



- optimize to axis-aligned
  - simple calculation
  - no floating point division unless intersection pt. required



Polygon Clipping

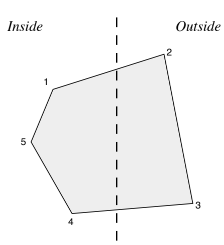


Input: polygon
Output: polygon(s)

Polygon Clipping: Sutherland-Hodgman

- convex clipping region
- clip against one edge at a time

P	C	Output
inside	inside	c
inside	outside	intersection point
outside	outside	-
outside	inside	intersection point; c



Polygon Clipping: Weiler-Atherton

- clipping region: concave, with holes
- polygon: concave (with holes)