Announcement

Project 2 has been posted.

Due: 23:59:59, Monday, November 7

Topics

Shading in OpenGL

From vertices to fragments

Examples

Example 1: vertex lighting

- Lighting is handled in vertex shader
- Color is computed for each vertex, then interpolated for each pixel

Example 2: fragment lighting

- Lighting is handled in fragment shader
- Color is computed for each fragment (potential pixel)

Example 1: Vertex Lighting Shaders (Vertex Shader)

```
// vertex shader
in vec4 vPosition;
in vec3 vNormal;
out vec4 color;  //vertex shade

// light and material properties
uniform vec4 AmbientProduct, DiffuseProduct,
SpecularProduct;
uniform vec4 LightPosition;
uniform float Shininess;

uniform mat4 ModelView;
uniform mat4 Projection;
```

Example 1: Vertex Lighting Shaders (Vertex Shader)

```
void main()
{
    // Transform vertex  position into eye coordinates
    vec3 pos = (ModelView * vPosition).xyz;
    vec3 L;
    // Compute the four vectors
    if (LightPosition.w!=0)//point light
      L = normalize((ModelView *LightPosition).xyz-pos);
    else //distant light
      L = normalize((ModelView * LightPosition).xyz);

    vec3 E = normalize( -pos );
    vec3 H = normalize( L + E );

    // Transform vertex normal into eye coordinates
    vec3 N = normalize( ModelView*vec4(vNormal, 0.0) ).xyz;
```

Example 1: Vertex Lighting Shaders (Vertex Shader)

```
// Compute terms in the illumination equation
    // Ambient term
    vec4 ambient = AmbientProduct;
    // Diffuse term
    float Kd = max( dot(L, N), 0.0 );
    vec4  diffuse = Kd*DiffuseProduct;
    // Specular term
    float Ks = pow( max(dot(N, H), 0.0), Shininess );
    vec4  specular = Ks * SpecularProduct;
 // discard the specular highlight if the light's behind
the vertex
    if( dot(L, N) < 0.0 )
        specular = vec4(0.0, 0.0, 0.0, 1.0);
    color = ambient + diffuse + specular;
    color.a = 1.0;
    gl_Position = Projection * ModelView * vPosition;
}
```

Example 1: Vertex Lighting Shaders (Fragment Shader)

```
// fragment shader

in vec4 color;

void main()
{
    gl_FragColor = color;
}
```

Example 1: Vertex Lighting Shaders (Application)

```
void init(){
                    …
   // Create a vertex array object
   GLuint vao;
   glGenVertexArrays( 1, &vao );
   glBindVertexArray( vao );

   // Create and initialize a buffer object
   GLuint buffer;
   glGenBuffers( 1, &buffer );
   glBindBuffer( GL_ARRAY_BUFFER, buffer );
   glBufferData( GL_ARRAY_BUFFER, sizeof(points) + sizeof(normals), NULL,
GL_STATIC_DRAW );

   glBufferSubData(GL_ARRAY_BUFFER,0,sizeof(points), points);
   glBufferSubData(GL_ARRAY_BUFFER,sizeof(points),sizeof(normals), normals );
```

Example 1: Vertex Lighting Shaders (Application)

```
…
// Load shaders and use the resulting shader program
    GLuint program = InitShader( "vshader", "fshader" );
    glUseProgram( program );

    // set up vertex arrays
    GLuint vPosition=glGetAttribLocation(program,"vPosition");
    glEnableVertexAttribArray( vPosition );
    glVertexAttribPointer( vPosition, 4, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(0) );

    GLuint vNormal = glGetAttribLocation( program, "vNormal" );
    glEnableVertexAttribArray( vNormal );
    glVertexAttribPointer( vNormal, 3, GL_FLOAT, GL_FALSE, 0,
                           BUFFER_OFFSET(sizeof(points)) );
```

Example 1: Vertex Lighting Shaders (Application)

```
// Initialize shader lighting parameters
point4 light_position( 0.0, 0.0, 1.0, 0.0 );
color4 light_ambient( 0.2, 0.2, 0.2, 1.0 );
color4 light_diffuse( 1.0, 1.0, 1.0, 1.0 );
color4 light_specular( 1.0, 1.0, 1.0, 1.0 );
```

lighting

```
color4 material_ambient( 1.0, 0.0, 1.0, 1.0 );
color4 material_diffuse( 1.0, 0.8, 0.0, 1.0 );
color4 material_specular( 1.0, 0.8, 0.0, 1.0 );
float  material_shininess = 100.0;
```

Material

Elementwise multiplication

```
color4 ambient_product = light_ambient * material_ambient;
color4 diffuse_product = light_diffuse * material_diffuse;
color4 specular_product = light_specular * material_specular;
```

Example 1: Vertex Lighting Shaders (Application)

```
glUniform4fv( glGetUniformLocation(program, "AmbientProduct"),1, ambient_product );
glUniform4fv( glGetUniformLocation(program, "DiffuseProduct"),1, diffuse_product );
glUniform4fv( glGetUniformLocation(program, "SpecularProduct"),1, specular_product );

glUniform4fv( glGetUniformLocation(program, "LightPosition"),1, light_position );
glUniform1f( glGetUniformLocation(program, "Shininess"),
                        material_shininess );

// Retrieve transformation uniform variable locations
ModelView = glGetUniformLocation( program, "ModelView" );
Projection = glGetUniformLocation( program, "Projection" );
glEnable( GL_DEPTH_TEST );
glShadeModel(GL_FLAT);
glClearColor( 1.0, 1.0, 1.0, 1.0 );
}
```

Mode available:

- GL_FLAT
- GL_SMOOTH (default)

Example 2: Fragment Lighting Shaders (Vertex Shader)

```
// vertex shader
in vec4 vPosition;
in vec3 vNormal;

// output values that will be interpolated per-fragment
out vec3 fN;
out vec3 fE;
out vec3 fL;

uniform mat4 ModelView;
uniform vec4 LightPosition;
uniform mat4 Projection;
```

Example 2: Fragment Lighting Shaders (Vertex Shader)

```
void main()
{
    vec3 pos = (ModelView*vPosition).xyz;

    fN = (ModelView*vec4(vNormal, 0.0)).xyz ;
    fE = -pos.xyz;
    if (LightPosition.w!=0) //point light
      fL = (ModelView*LightPosition).xyz - pos;
    else //distant light
      fL = (ModelView*LightPosition).xyz;

    gl_Position = Projection*ModelView*vPosition;
}
```

Example 2: Fragment Lighting Shaders (Fragment Shader)

```
// fragment shader

// per-fragment interpolated values from the vertex shader
in vec3 fN;
in vec3 fL;
in vec3 fE;

uniform vec4 AmbientProduct, DiffuseProduct,SpecularProduct;
uniform mat4 ModelView;
uniform vec4 LightPosition;
uniform float Shininess;
```

Example 2: Fragment Lighting Shaders (Fragment Shader)

```
void main()
{
    // Normalize the input lighting vectors

   vec3 N = normalize(fN);
    vec3 E = normalize(fE);
    vec3 L = normalize(fL);

    vec3 H = normalize( L + E );
    vec4 ambient = AmbientProduct;
```

Example 2: Fragment Lighting Shaders (Fragment Shader)

```
    float Kd = max(dot(L, N), 0.0);
    vec4 diffuse = Kd*DiffuseProduct;

    float Ks = pow(max(dot(N, H), 0.0), Shininess);
    vec4 specular = Ks*SpecularProduct;

    // discard the specular highlight if the light's
behind the vertex
    if( dot(L, N) < 0.0 )
      specular = vec4(0.0, 0.0, 0.0, 1.0);

    gl_FragColor = ambient + diffuse + specular;
    gl_FragColor.a = 1.0;
}
```

Per-vertex Lighting vs Per-fragment Lighting

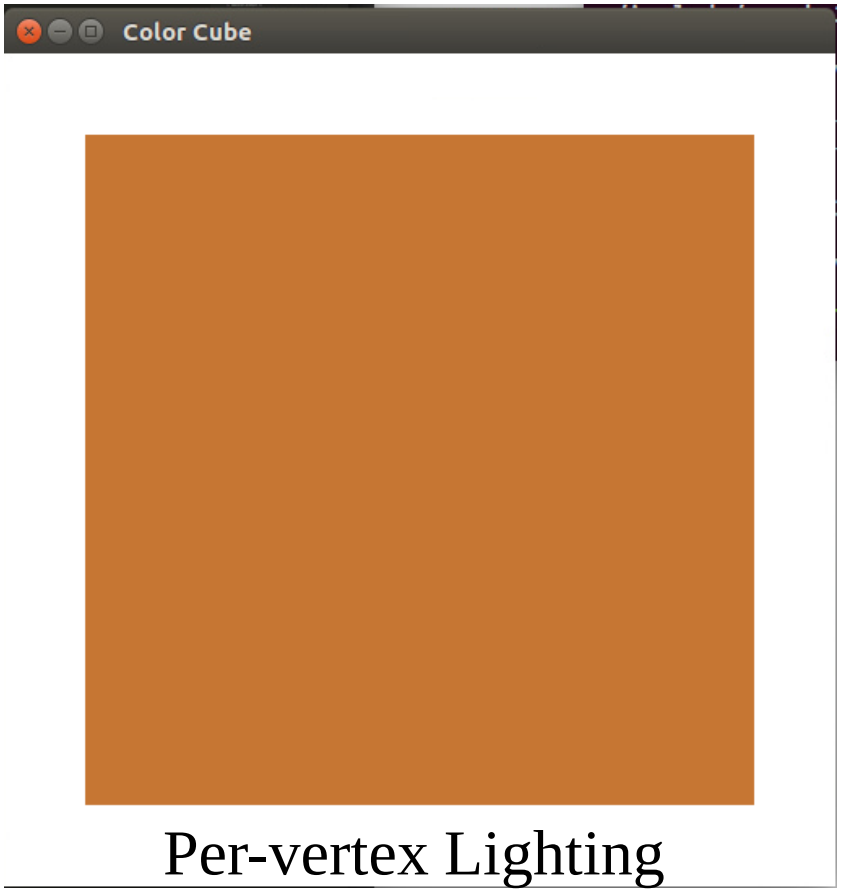
Per-vertex lighting

- Lighting is handled in vertex shader
- Color is computed for each vertex, then interpolated for each pixel
- Efficient, but rough

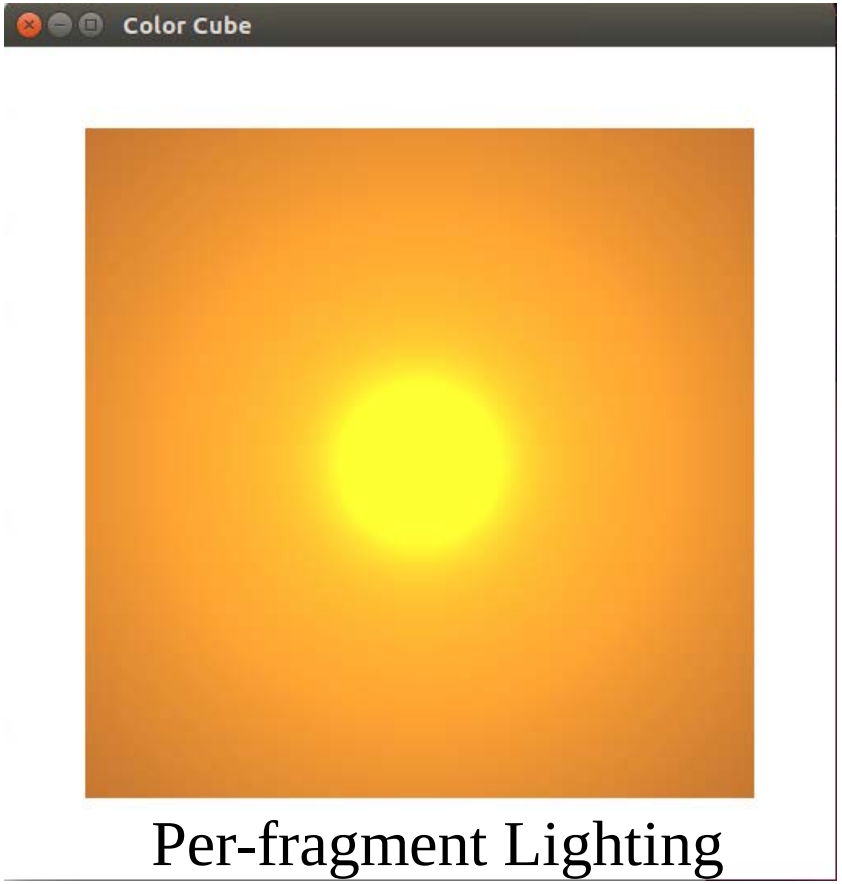
Per-fragment lighting

- Lighting is handled in fragment shader
- Color is computed for each fragment (potential pixel)
- Sophisticated, but slow

Per-vertex Lighting vs Per-fragment Lighting



Per-vertex Lighting



Per-fragment Lighting

From Vertices to Fragments

Assign a color to every pixel

Pass every object through the system

At end of the geometric pipeline, vertices have been assembled into primitives

Must clip out primitives that are outside the view frustum

- Algorithms based on representing primitives by lists of vertices

Must find which pixels can be affected by each primitive

- Fragment generation
- Rasterization or scan conversion

Required Tasks

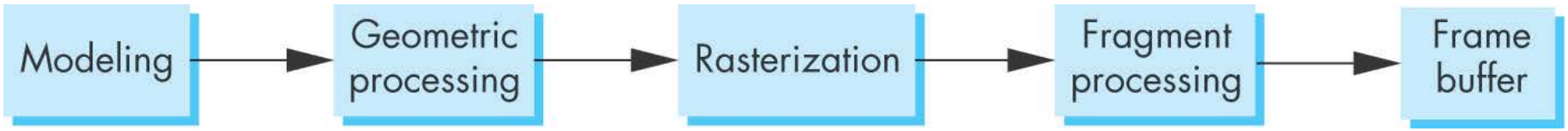
Modeling

Geometric processing

Rasterization

Fragment processing including

- Hidden surface removal
- Antialiasing



Modeling

Produce geometric objects in application

Tasks:

- Geometric modeling
- Clipping

Geometric Processing

Determine which objects can be shown and assign colors to vertices

- Projection
  - object frame-camera frame-clip coordinates
- Primitive assembly
- Clipping
  - Generate new primitives in the view volume
  - Normalized device coordinates
- Perspective division
- shading

Rasterization or Scan Conversion

Generate a set of fragment – x, y coordinates of the vertices in *units of window coordinates* in the frame buffer, e.g.

- Fragments to approximate a line segment
- Pixels to form a polygon

Most of graphics systems do not have special rasterization algorithms

Fragment Processing

Simplest case:

Assign a color to each fragment and place it in the buffer corresponding to the fragment

More complicated cases:

- Combine texture colors and fragment colors in the color buffer
- Hidden surface removal
- Blend color for translucent objects
- Reducing jaggedness or aliasing

Clipping and Visibility

Clipping has much in common with hidden-surface removal

In both cases, we are trying to remove objects that are not visible to the camera

Often we can use visibility or occlusion testing early in the process to eliminate as many polygons as possible before going through the entire pipeline

Clipping

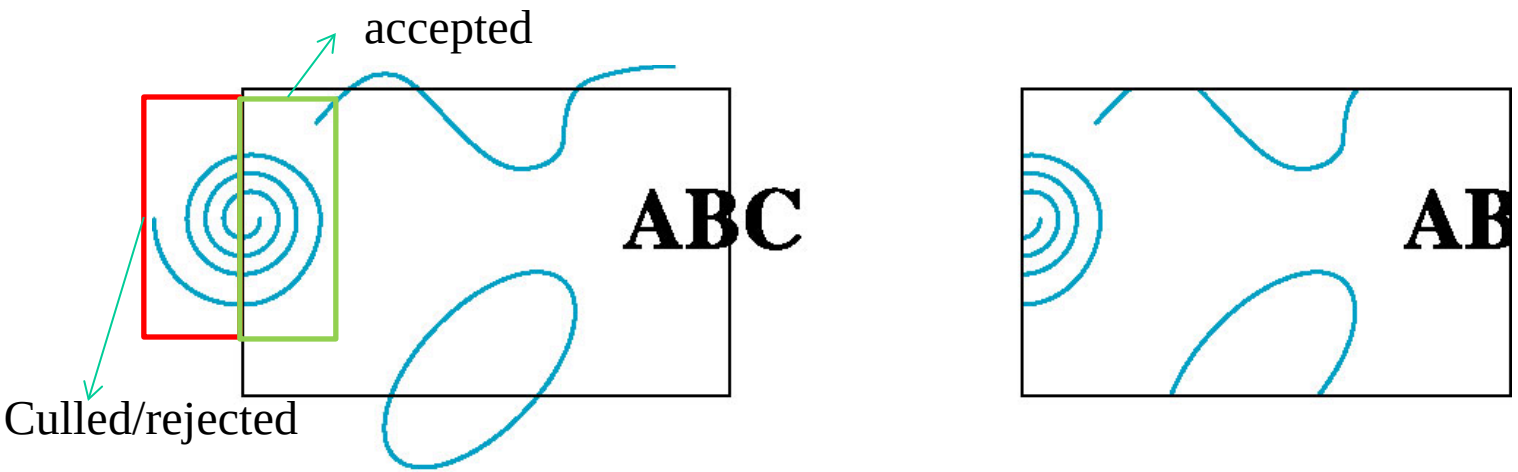
2D against clipping window → Projection plane mapped to viewport

3D against clipping volume

Easy for line segments and polygons

Hard for curves and text

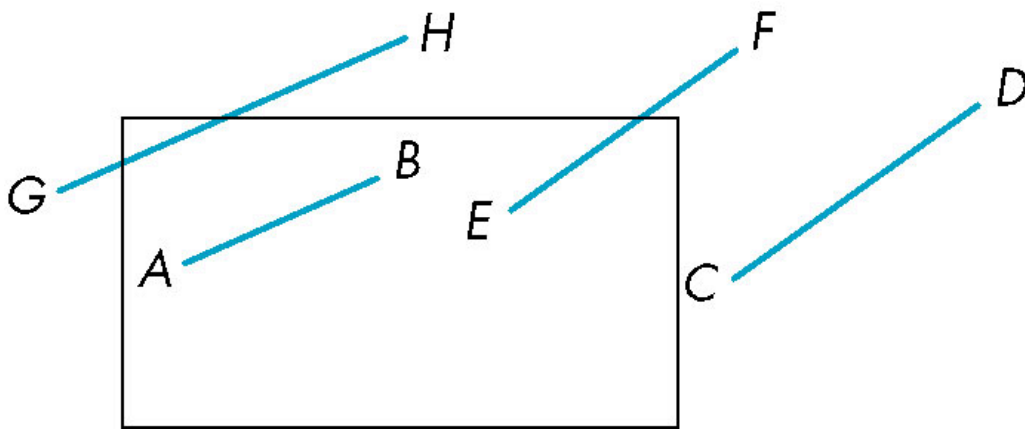
- Convert to lines and polygons first



Clipping 2D Line Segments

Brute force approach: compute intersections with all sides of clipping window

- Inefficient: one division per intersection

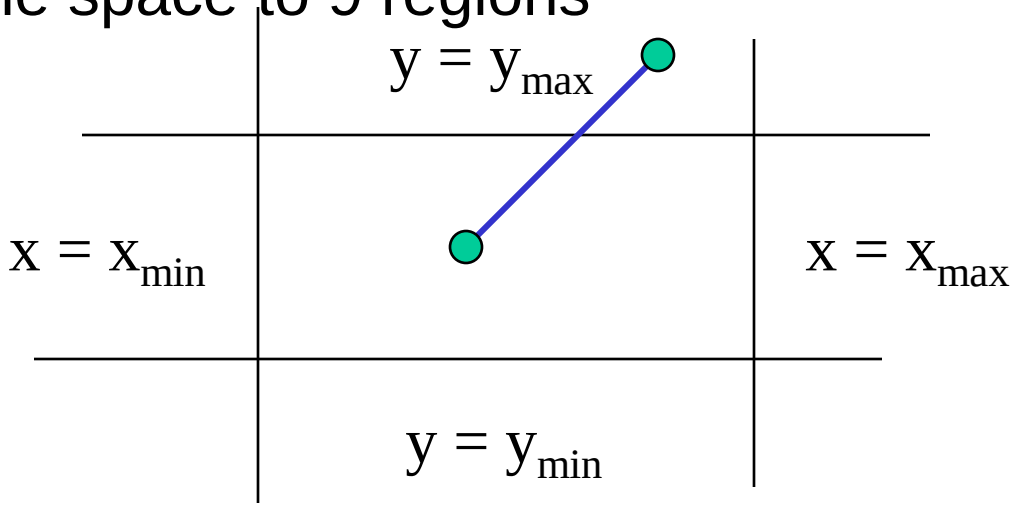


Cohen-Sutherland Algorithm

Idea: eliminate as many cases as possible without computing intersections

Start with four lines that determine the sides of the clipping window

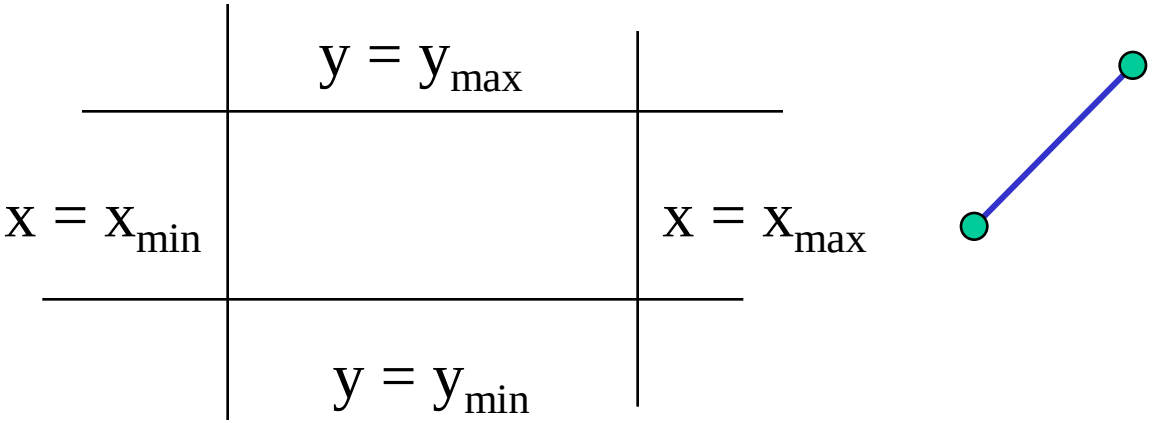
Divide the space to 9 regions



Four Cases Considering Endpoints

Case 1: both endpoints of line segment inside all four lines

- Draw (accept) line segment as is



Case 2: both endpoints outside all lines and on same side of a line

- Discard (reject) the line segment

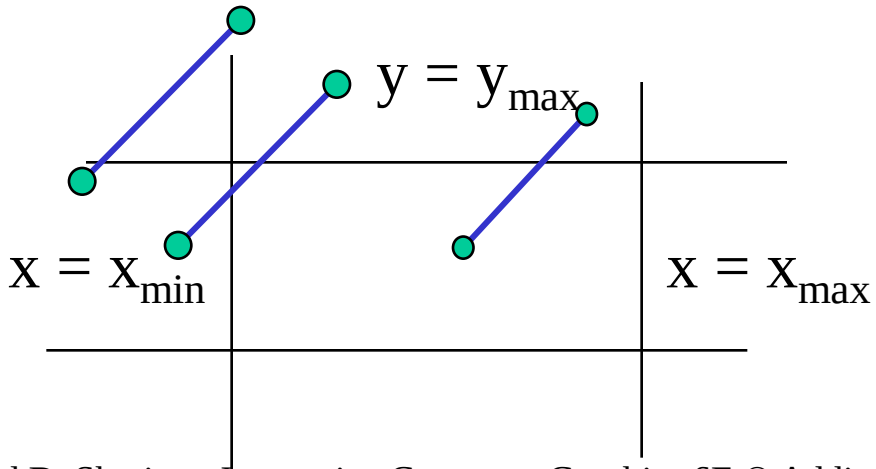
The Cases

Case 3: One endpoint inside, one outside

- Must do at least one intersection

Case 4: Both outside

- May have part inside
- Must do at least one intersection

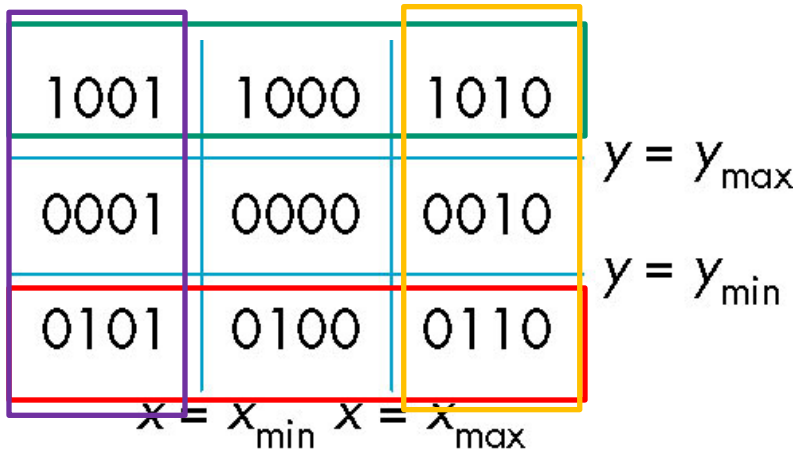


Defining Outcodes

For each endpoint, define an outcode

$b_0b_1b_2b_3$

$b_0 = 1$ if $y > y_{max}$, 0 otherwise
$b_1 = 1$ if $y < y_{min}$, 0 otherwise
$b_2 = 1$ if $x > x_{max}$, 0 otherwise
$b_3 = 1$ if $x < x_{min}$, 0 otherwise



Outcodes divide space into 9 regions

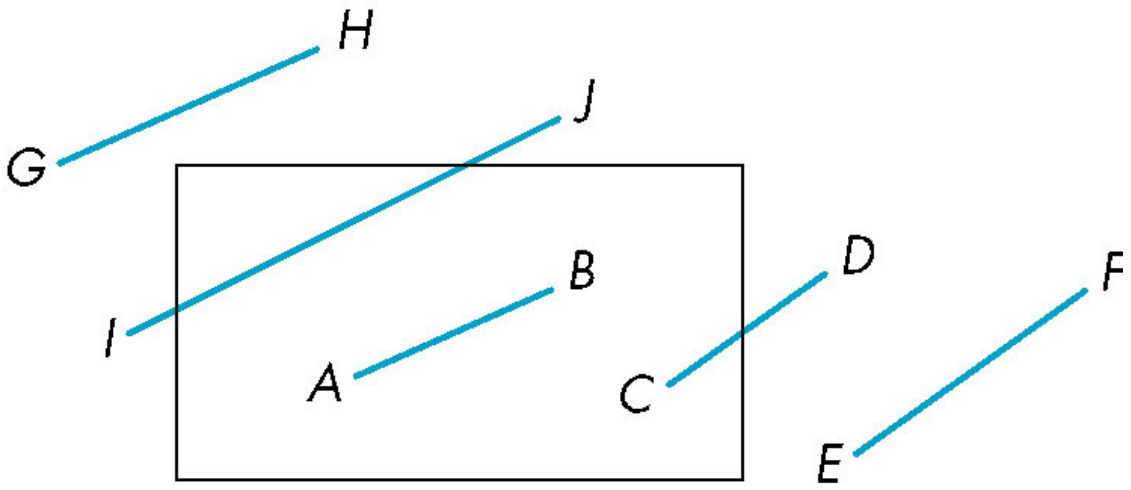
Computation of outcode requires at most 4 subtractions

Using Outcodes

Consider the 5 cases below

AB: outcode(A) = outcode(B) = 0

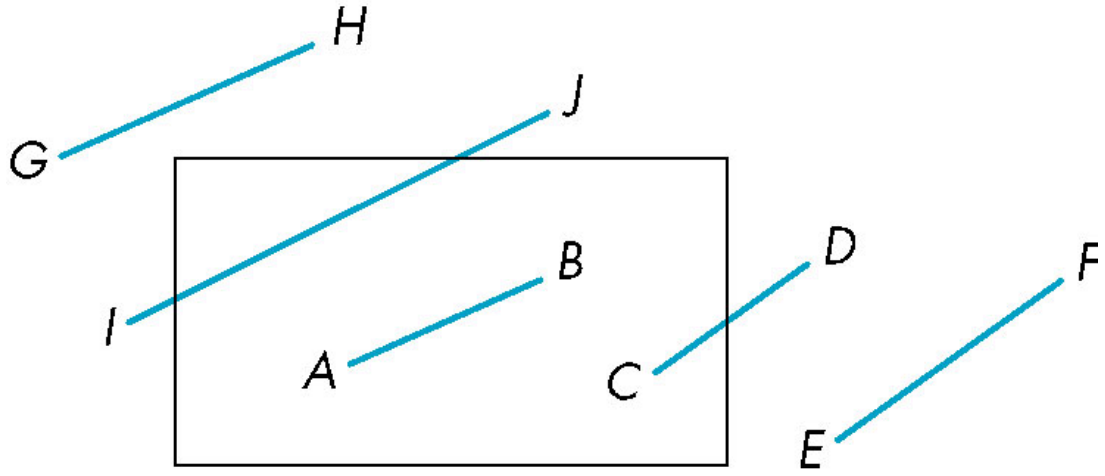
- Accept line segment



Using Outcodes

CD: outcode (C) = 0, outcode(D) ≠ 0

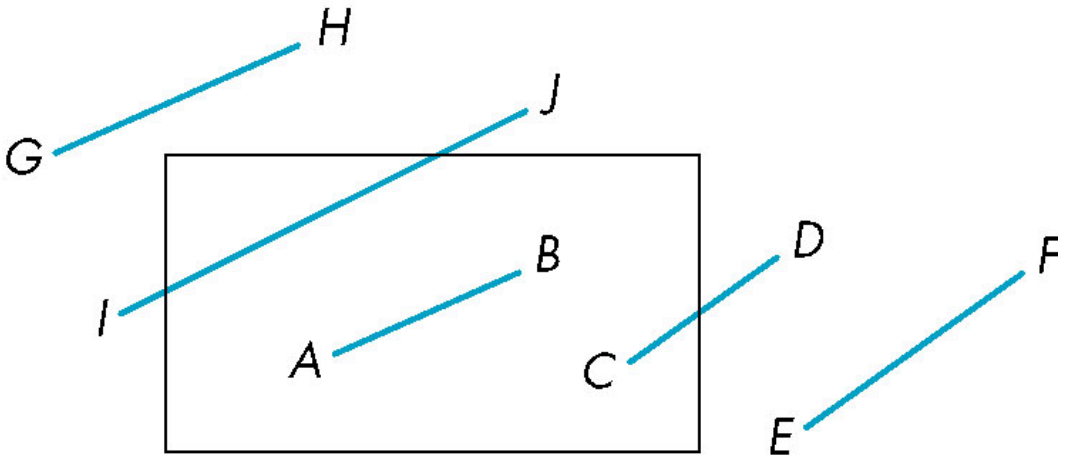
- Compute intersection
- Location of 1 in outcode(D) determines which edge to intersect with
- Note if there were 2 ones in outcode, we might have to do two intersections



Using Outcodes

EF: outcode(E) AND outcode(F) (bitwise) ≠ 0

- Both outcodes have a 1 bit in the same place
- Line segment is outside of corresponding side of clipping window
- reject



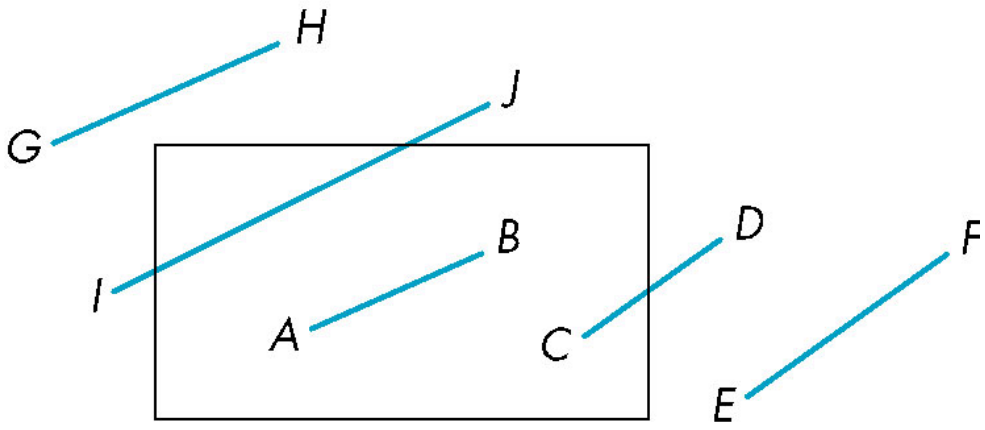
Using Outcodes

GH and IJ have same outcodes,

- outcode(G)≠0, outcode(H)≠0
- outcode(G) AND outcode(H) =0
- Shorten line segment by intersecting with one of sides of window

Compute outcode of intersection (new endpoint of shortened line segment)

Reexecute algorithm



Efficiency

In many applications, the clipping window is small relative to the size of the entire data base

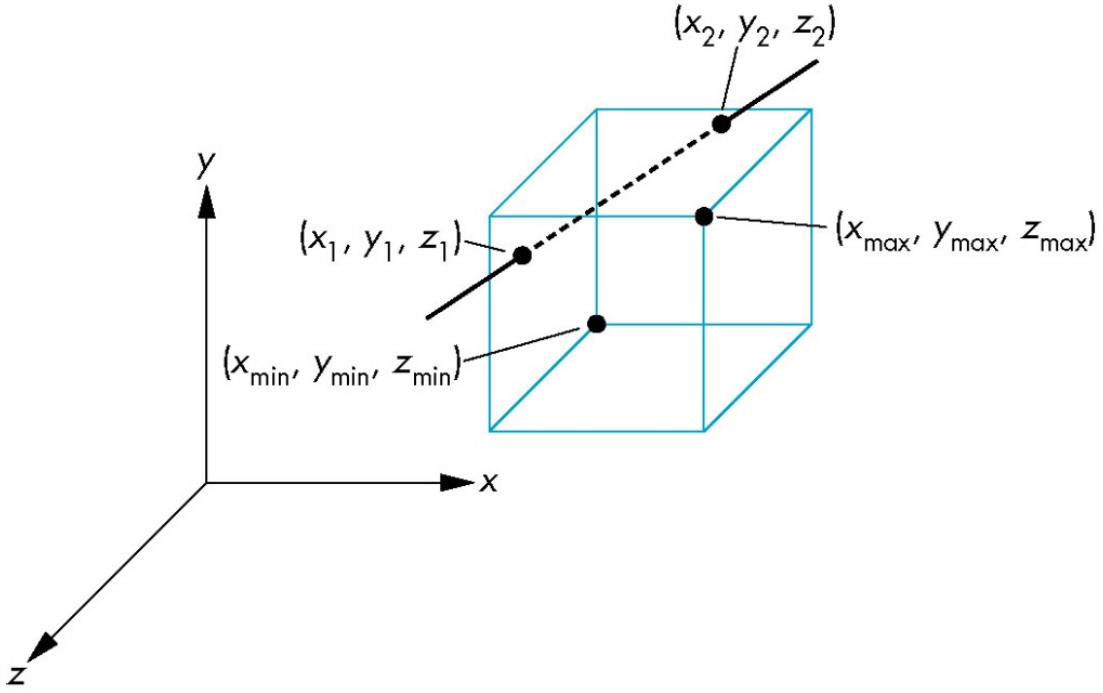
- Most line segments are outside one or more side of the window and can be eliminated based on their outcodes

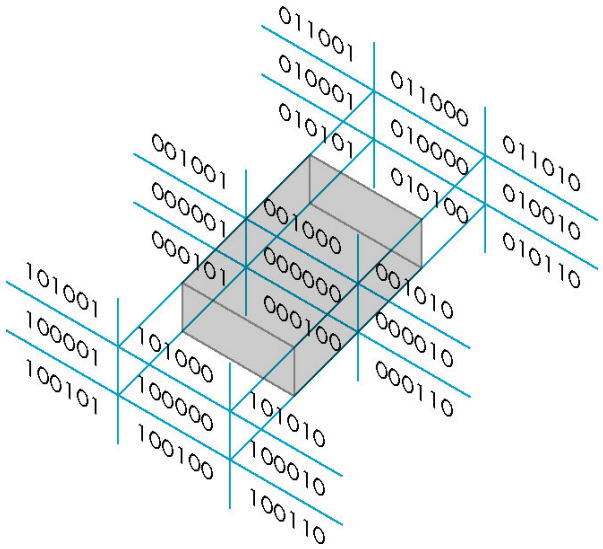
Inefficient when code has to be reexecuted for line segments that must be shortened in more than one step

Cohen Sutherland in 3D

Use 6-bit outcodes

When needed, clip line segment against planes



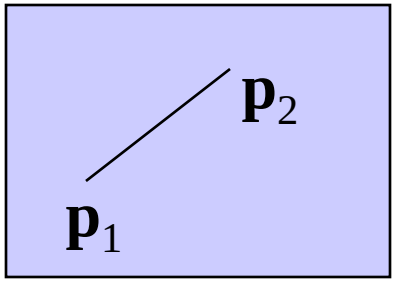


Liang-Barsky Clipping

Consider the parametric form of a line segment

$$\mathbf{p}(\alpha) = (1-\alpha)\mathbf{p}_1 + \alpha\mathbf{p}_2 \quad 1 \geq \alpha \geq 0$$

$0 \leq \alpha \leq 1$, points within the line segment



We can distinguish between the cases by looking at the ordering of the values of α where the line determined by the line segment crosses the lines that determine the window

Liang-Barsky Clipping

When the line is not parallel to a side of the window

In (a): $\alpha_4 > \alpha_3 > \alpha_2 > \alpha_1$

- Intersect right, top, left, bottom: shorten

In (b): $\alpha_4 > \alpha_2 > \alpha_3 > \alpha_1$

- Intersect right, left, top, bottom: reject