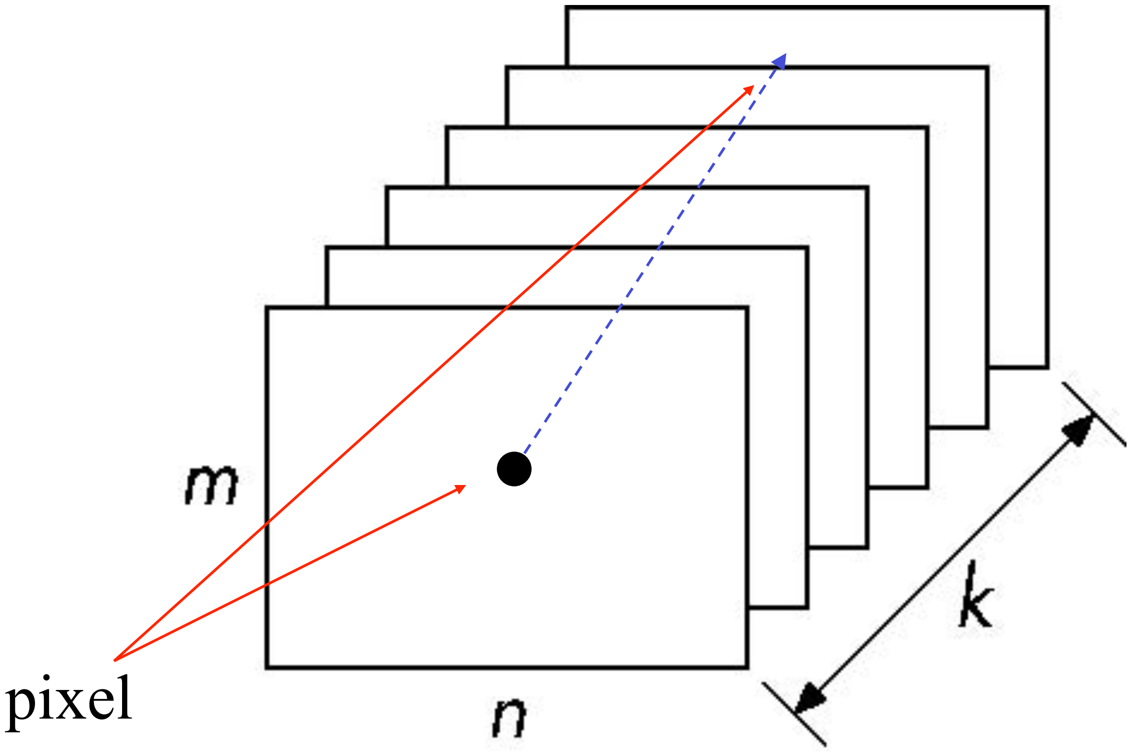
Pixels and Buffers

CS 432 Interactive Computer Graphics
Prof. David E. Breen
Department of Computer Science

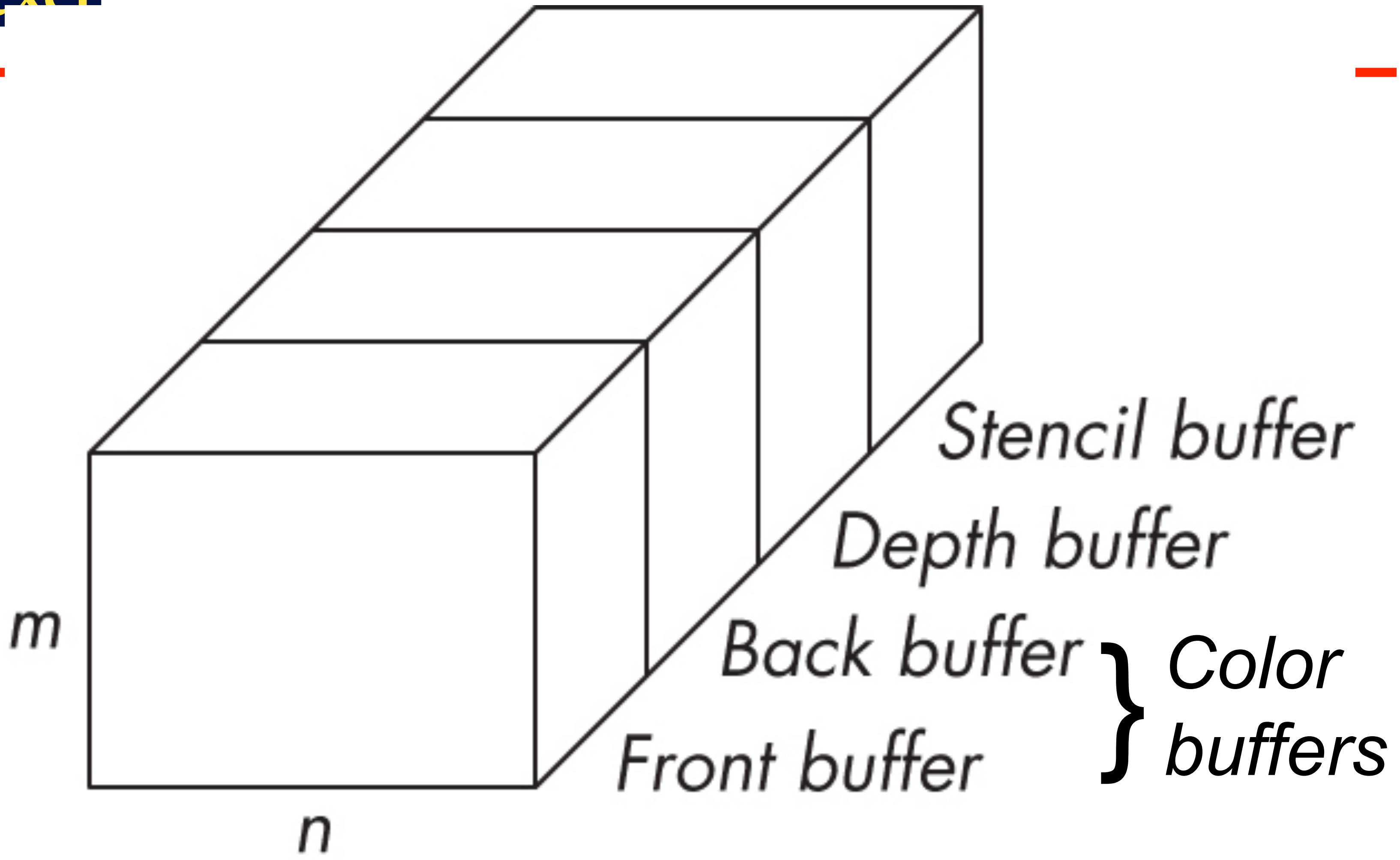
Image Buffer

Define a buffer by its spatial resolution (n x m) and its depth (or precision) k, the number of bits/pixel



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

OpenGL Frame Buffer

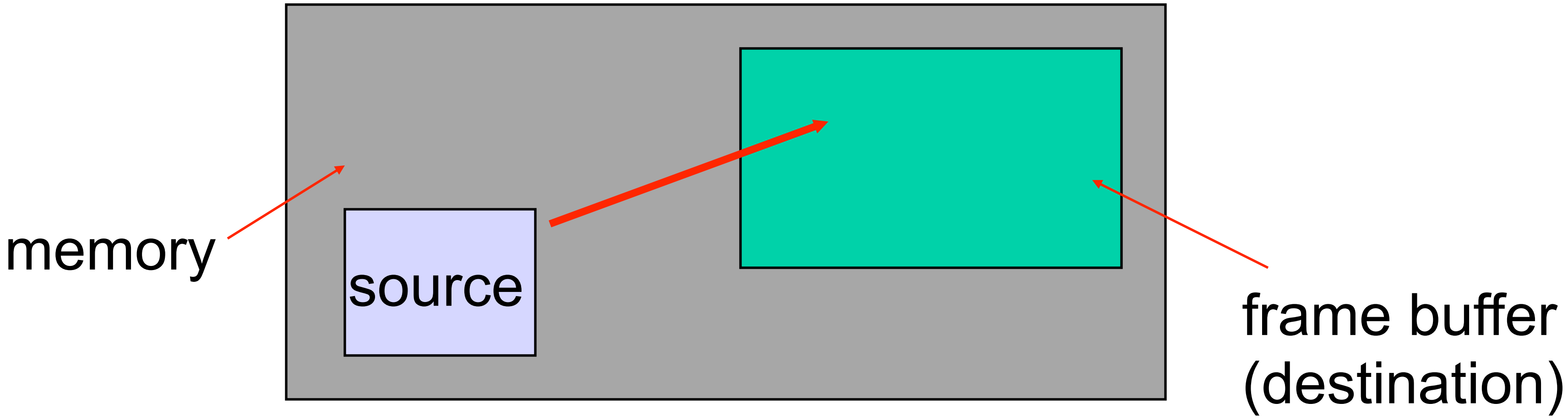


OpenGL Image Buffers

- Color buffers can be displayed
  - Front & Back
  - Auxiliary (off-screen)
  - Stereo
- Depth
- Stencil
  - Holds masks
  - Restricts drawing to portion of screen
- Most RGBA buffers 8 bits per component
- Latest are floating point (IEEE)

Writing in Buffers

- Conceptually, we can consider all of memory as a large two-dimensional array of pixels
- We read and write rectangular block of pixels
  - *Bit block transfer* (*bitblt*) *operations*
- The frame buffer is part of this memory



writing into frame buffer

Clearing Buffers

- A clear (default) value may be set for each buffer
  - `glClearColor()`
  - `glClearDepth()`
  - `glClearDepthf()`
  - `glClearStencil()`

- **`glClear(Glbitfield mask)`**
- Clears the specified buffer

Masking Buffers

- A buffer may be *mask'*ed, i.e. enabled or disabled
- **`glColorMask()`**
- **`glColorMaski()`**
  - Color buffer **`i`**
- **`glDepthmask()`**
- **`glStencilMask()`**
- **`glStencilMaskSeparate()`**
  - Stencil specific sides (front & back) of triangles

Fragment Tests and Operations

- After the fragment shader is executed a series of tests and operations are performed on the fragment

- Determine how and whether a fragment color is drawn into the frame buffer

Fragment Tests and Operations

- Tests and operations are performed in the following order
  - Scissor test
  - Multisample fragment operations
  - Stencil test
  - Depth test
  - Blending
  - Dithering
  - Logical operations
- On/off **`glEnable(), glDisable()`**

Pixel Tests

- Scissor
  - Only draw in a rectangular portion of screen
  - **`glScissor()`** – Specify rectangle
  - Default rectangle matches window
- Depth
  - Draw based on depth value and comparison function
  - **`glDepthFunc()`** – Specify comparison function
  - Default is **`GL_LESS`**

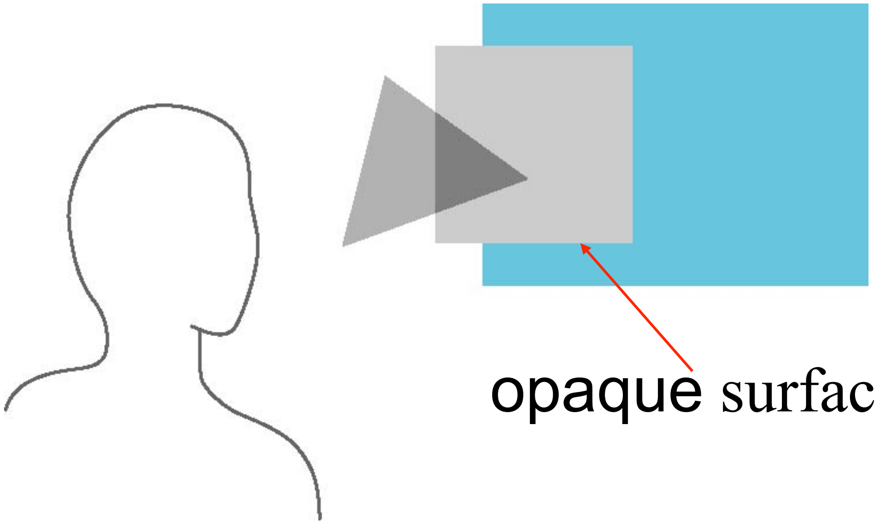
Pixel Tests

- Stencil
  - Draw based on values in stencil buffer, if available and enabled
  - Used for drawing into an irregular region of color buffer
  - **`glStencilFunc()`** – Specifies comparison function, reference value and mask
  - **`glStencilOp()`** – Specifies how fragments can modify stencil buffer
  - Used for reflections, capping and stippling

Opacity and Transparency

- Opaque surfaces permit no light to pass through
- Transparent surfaces permit all light to pass
- Translucent surfaces pass some light

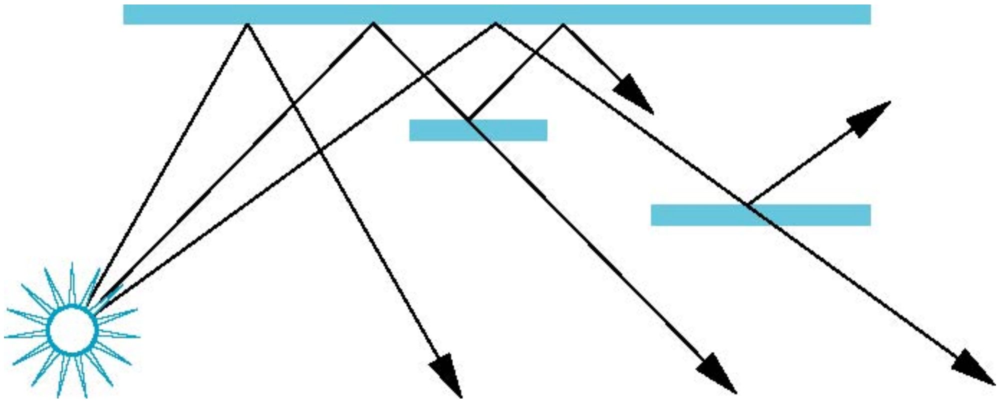
translucency = 1 – opacity (α)



opaque surface α =1

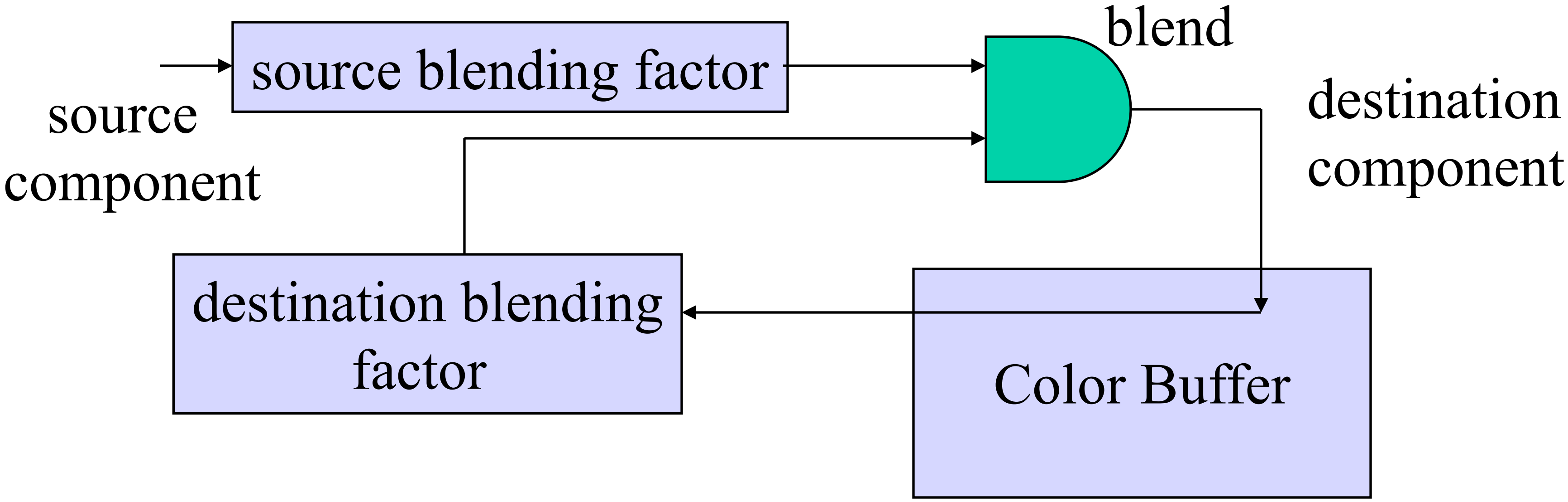
Physical Models

- Dealing with translucency in a physically correct manner is difficult due to
  - the complexity of the internal interactions of light and matter
  - Using a pipeline renderer



Writing Model for Blending

- Use A component of RGBA (or RGBα) color to store opacity
- During rendering we can expand our writing model to use RGBA values



blend

source component

source blending factor

destination component

destination blending factor

Color Buffer

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

Blending Equation

- We can define source and destination blending factors for each RGBA component

$$\mathbf{s} = [s_r, s_g, s_b, s_\alpha]$$

$$\mathbf{d} = [d_r, d_g, d_b, d_\alpha]$$

Suppose that the source and destination colors are

$$\mathbf{b} = [b_r, b_g, b_b, b_\alpha]$$

$$\mathbf{c} = [c_r, c_g, c_b, c_\alpha]$$

Blend as

$$\mathbf{c}' = [b_r\, s_r + c_r\, d_r, b_g\, s_g + c_g\, d_g\,, b_b\, s_b + c_b\, d_b\,, b_\alpha\, s_\alpha + c_\alpha\, d_\alpha\,]$$

OpenGL Blending

- Must enable blending and set source and destination factors

```
glEnable(GL_BLEND)
glBlendFunc(source_factor,
                              destination_factor)
```

- Only certain factors supported
  - `GL_ZERO, GL_ONE`
  - `GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA`
  - `GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA`
  - See Redbook for complete list

Example

- Suppose that we start with the opaque background color $(R_0,G_0,B_0,1)$
  - This color becomes the initial destination color
- We now want to blend in a translucent polygon with color (R_1,G_1,B_1,α_1)
- Select **`GL_SRC_ALPHA`** and **`GL_ONE_MINUS_SRC_ALPHA`** as the source and destination blending factors

$$R'_1 = \alpha_1 R_1 + (1- \alpha_1) R_{0,} \ldots\ldots$$

- Note this formula is correct if polygon is either opaque or transparent

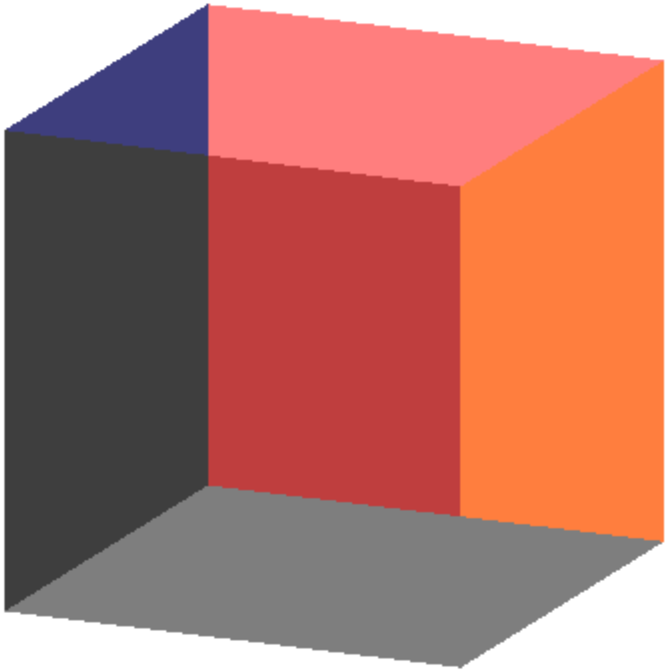
E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

Clamping and Accuracy

- All the components (RGBA) are clamped and stay in the range (0,1)
- However, in a typical system, RGBA values are only stored to 8 bits
  - Can easily loose accuracy if we add many components together
  - Example: add together n images
    - Divide all color components by n to avoid clamping
    - Blend with source factor = 1, destination factor = 1
    - But division by n loses bits

Order Dependency

- Is this image correct?
  - Probably not
  - Polygons are rendered in the order they pass down the pipeline
  - Blending functions are order dependent

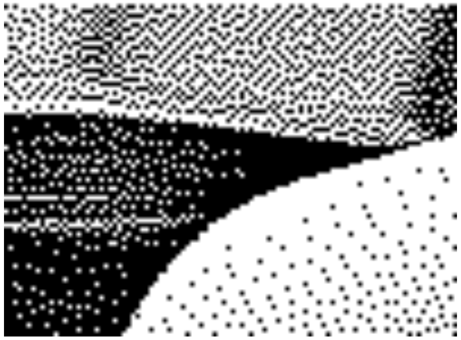


Opaque and Translucent Polygons

- Suppose that we have a group of polygons some of which are opaque and some translucent
- How do we use hidden-surface removal?
- Opaque polygons block all polygons behind them and affect the depth buffer
- Translucent polygons should not affect depth buffer
  - Render with **`glDepthMask(GL_FALSE)`** which makes depth buffer read-only
- Sort polygons first to remove order dependency!
- Draw back to front

Dithering and Logical Operations

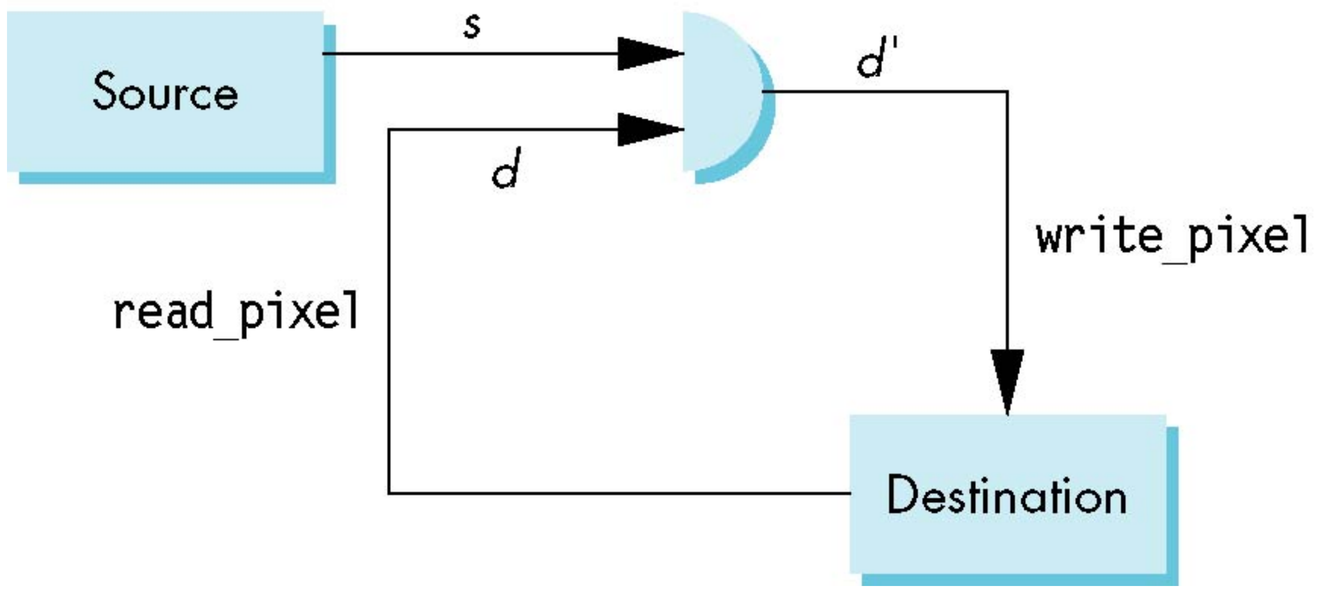
- Dithering
  - On some systems with limited color resolution dithering may be enabled (**`GL_DITHER`**)
  - System/hardware-dependent



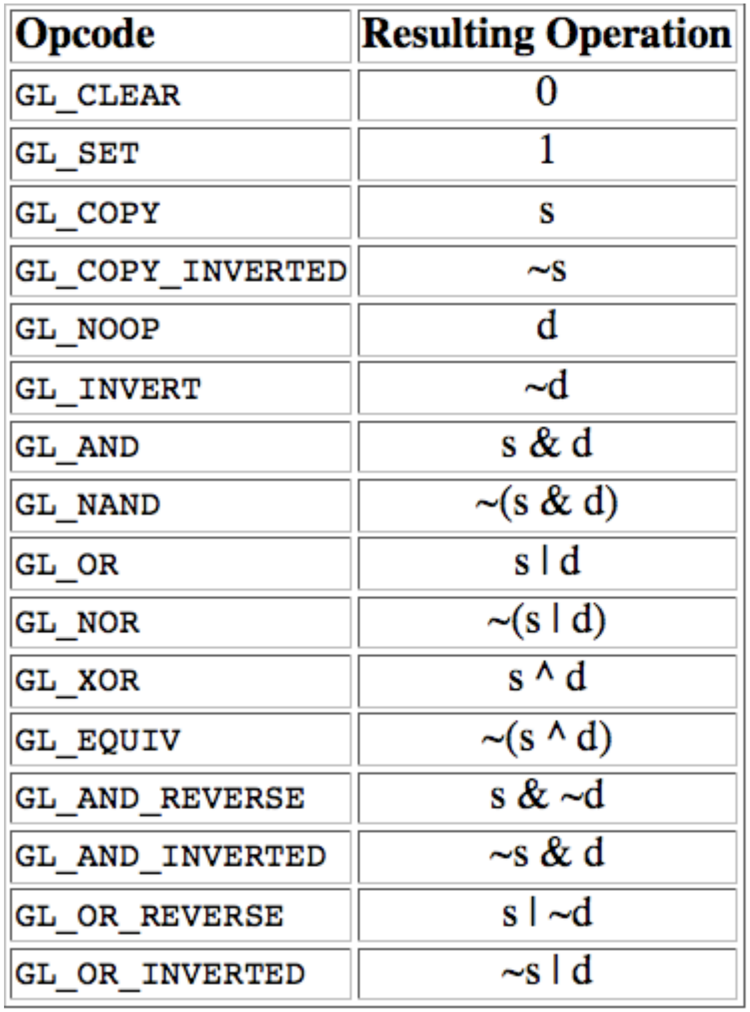
- Final operation combines fragment color with pixel color with a logical operator

Writing Model for Logical Operations

Read destination pixel before writing source



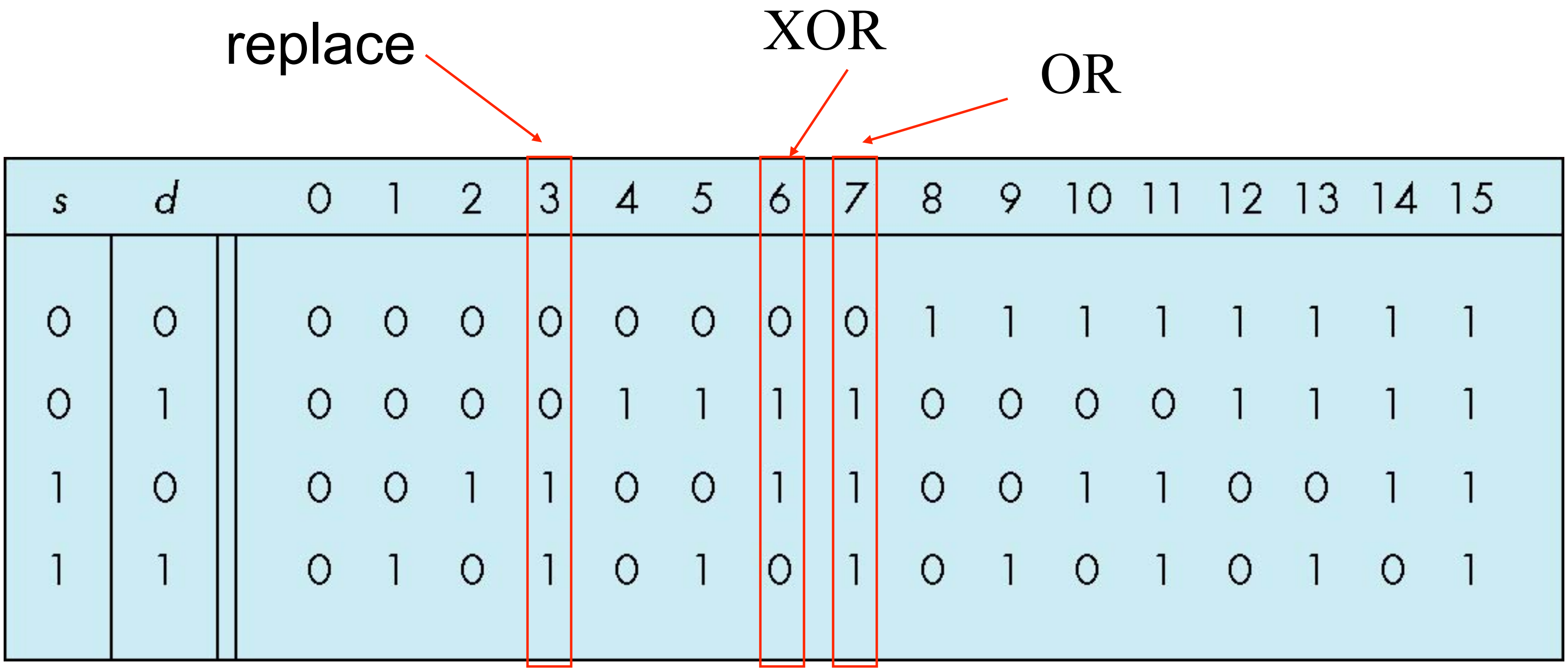
Logical Pixel Operations



Opcode	Resulting Operation
GL_CLEAR	0
GL_SET	1
GL_COPY	s
GL_COPY_INVERTED	~s
GL_NOOP	d
GL_INVERT	~d
GL_AND	s & d
GL_NAND	~(s & d)
GL_OR	s | d
GL_NOR	~(s | d)
GL_XOR	s ^ d
GL_EQUIV	~(s ^ d)
GL_AND_REVERSE	s & ~d
GL_AND_INVERTED	~s & d
GL_OR_REVERSE	s | ~d
GL_OR_INVERTED	~s | d

Bit Writing Modes

- Source and destination bits are combined bitwise
- 16 possible functions (one per column in table)



s	*d*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

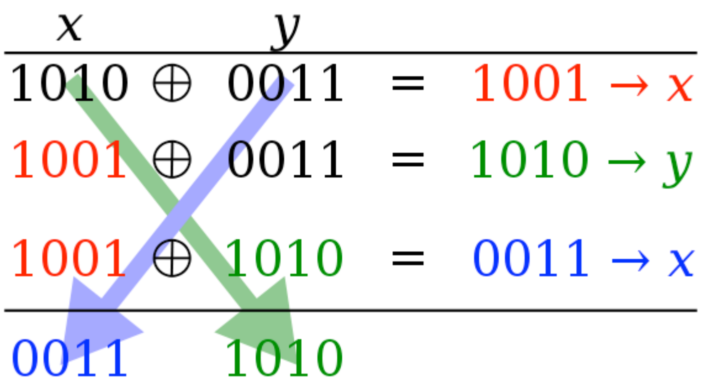
XOR mode

- Recall from Chapter 3 that we can use XOR by enabling logic operations and selecting the XOR write mode
- XOR is especially useful for swapping blocks of memory such as menus that are stored off screen

If S represents screen and M represents a menu the sequence

$$\mathrm{S} \leftarrow \mathrm{S} \oplus \mathrm{M}$$
$$\mathrm{M} \leftarrow \mathrm{S} \oplus \mathrm{M}$$
$$\mathrm{S} \leftarrow \mathrm{S} \oplus \mathrm{M}$$

swaps the S and M

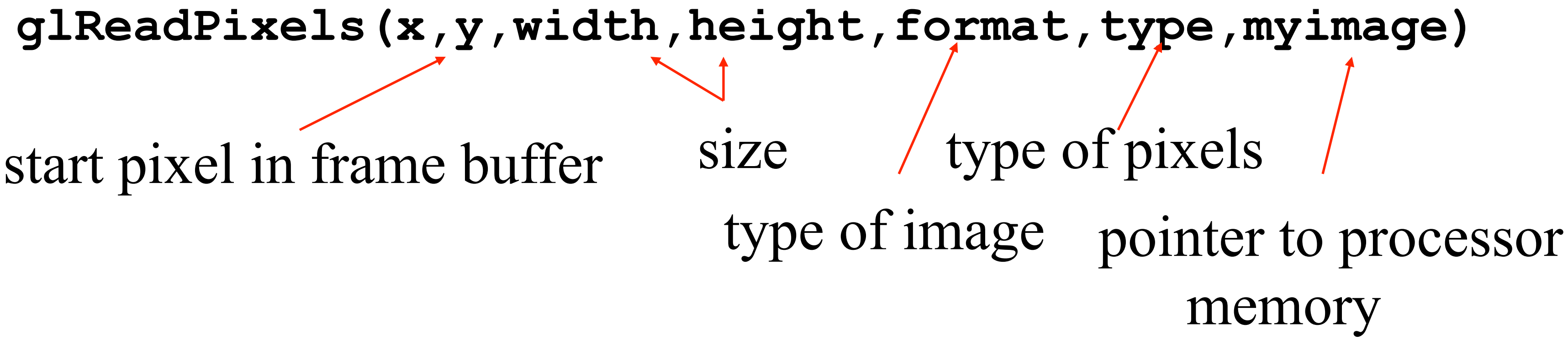


$$\begin{array}{ccccc} x & & y & & \\ \hline 1010 & \oplus & 0011 & = & 1001 \rightarrow x \\ 1001 & \oplus & 0011 & = & 1010 \rightarrow y \\ 1001 & \oplus & 1010 & = & 0011 \rightarrow x \\ \hline 0011 & & 1010 & & \end{array}$$

Buffer Selection

- OpenGL can read from any of the buffers (front, back, depth, stencil)
- Default to the back buffer
- Change with **`glReadBuffer`**
- Note that format of the pixels in the frame buffer is different from that of processor memory and these two types of memory reside in different places
    - Need packing and unpacking
    - Reading can be slow
- Drawing through texture functions

OpenGL Pixel Functions



```
glReadPixels(x,y,width,height,format,type,myimage)
```

start pixel in frame buffer

size

type of image

type of pixels

pointer to processor memory

```
GLubyte myimage[512][512][3];
glReadPixels(0,0, 512, 512, GL_RGB,
      GL_UNSIGNED_BYTE, myimage);
```

Formats & Types

- `GL_RGB`
- `GL_RGBA`
- `GL_RED`
- `GL_GREEN`
- `GL_BLUE`
- `GL_ALPHA`
- `GL_DEPTH_COMPONENT`
- `GL_LUMINANCE`
- `GL_LUMINANCE_ALPHA`
- `GL_COLOR_INDEX`
- `GL_STENCIL_INDEX`

- `GL_UNSIGNED_BYTE`
- `GL_BYTE`
- `GL_BITMAP`
- `GL_UNSIGNED_SHORT`
- `GL_SHORT`
- `GL_UNSIGNED_INT`
- `GL_INT`
- `GL_FLOAT`
- `GL_UNSIGNED_BYTE_3_3_2`
- `GL_UNSIGNED_INT_8_8_8_8`
- etc.

Deprecated Functionality

- glDrawPixels
- glCopyPixels
- glBitMap
- Replace by use of texture functionality, glBltFrameBuffer, frame buffer objects

Render to Texture

- GPUs now include a large amount of texture memory that we can write into
- Advantage: fast (not under control of window system)
- Using frame buffer objects (FBOs) we can render into texture memory instead of the frame buffer and then read from this memory
  - Image processing
  - GPGPU

Frame Buffer Objects

- Frame buffer useful for off-screen rendering, moving data between buffers and updating texture maps
- Attach renderbuffers to minimize data copies and optimize performance
- The window-system-provided buffers can never be associated with a framebuffer object

Frame Buffer Object

- **`glGenFramebuffers()`** – Allocate unused framebuffer object ids
- **`glBindFramebuffer()`** – Allocate storage for framebuffer and specifies read/write status
- Frame buffer parameters normally determined by its attachments

Renderbuffers

- Does memory management of formatted image data
- **`glGenRenderbuffers()`** – Allocate unused renderbuffer ids
- **`glBindRenderbuffer()`** – Sets state info to defaults and allows state info to be modified
- **`glRenderbufferStorage()`** – Allocate storage and specify image format

Attaching a Renderbuffer

- **`glFramebufferRenderbuffer()`** – Attaches a renderbuffer to a framebuffer. Specifies buffer type
- Type can be color, depth or stencil

Moving Pixels Around

- **`glDrawBuffer()`** – Specifies color buffer enabled for writing/clearing
- **`glReadBuffer()`** – Specifies color buffer enabled for source of reading
- **`glBlitFramebuffer()`** – Copies pixels from one buffer to another
- **`glReadPixels()`** – Copies pixels from the “read” buffer into an array

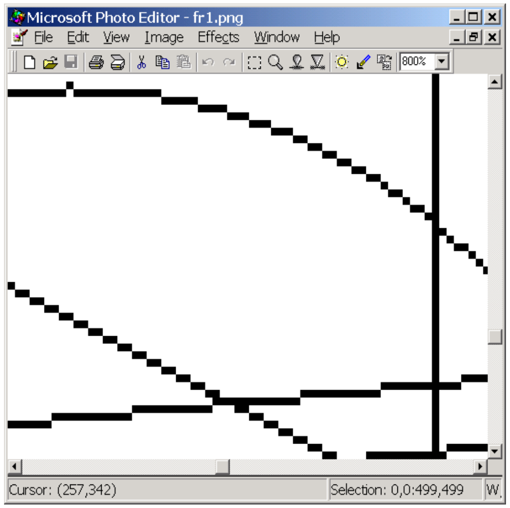
- Go to RenderBuffer.txt

- This is example 4.11 in the Red Book, 8th edition

Buffer Applications

Anti-aliasing

- Aliasing – artifacts produced from inadequate sampling
  - Jagged edges
  - Missing thin objects/features



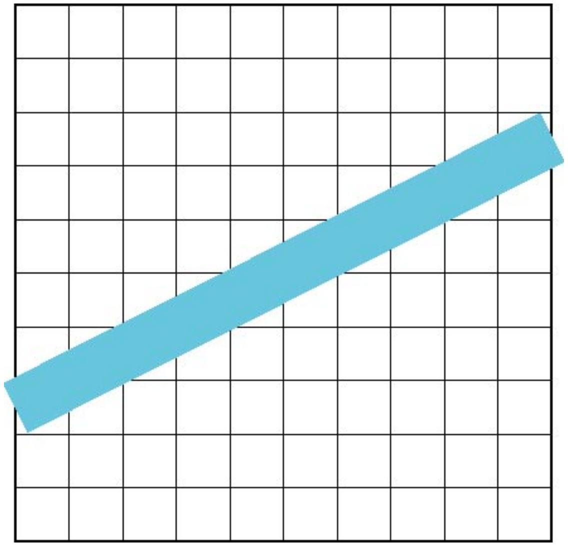
Cursor: (257,342)

Selection: 0,0:499,499

- Anti-aliasing – removing artifacts via super-sampling, filtering, blurring, smoothing
- OpenGL offers a number of ways to perform anti-aliasing

Line Aliasing

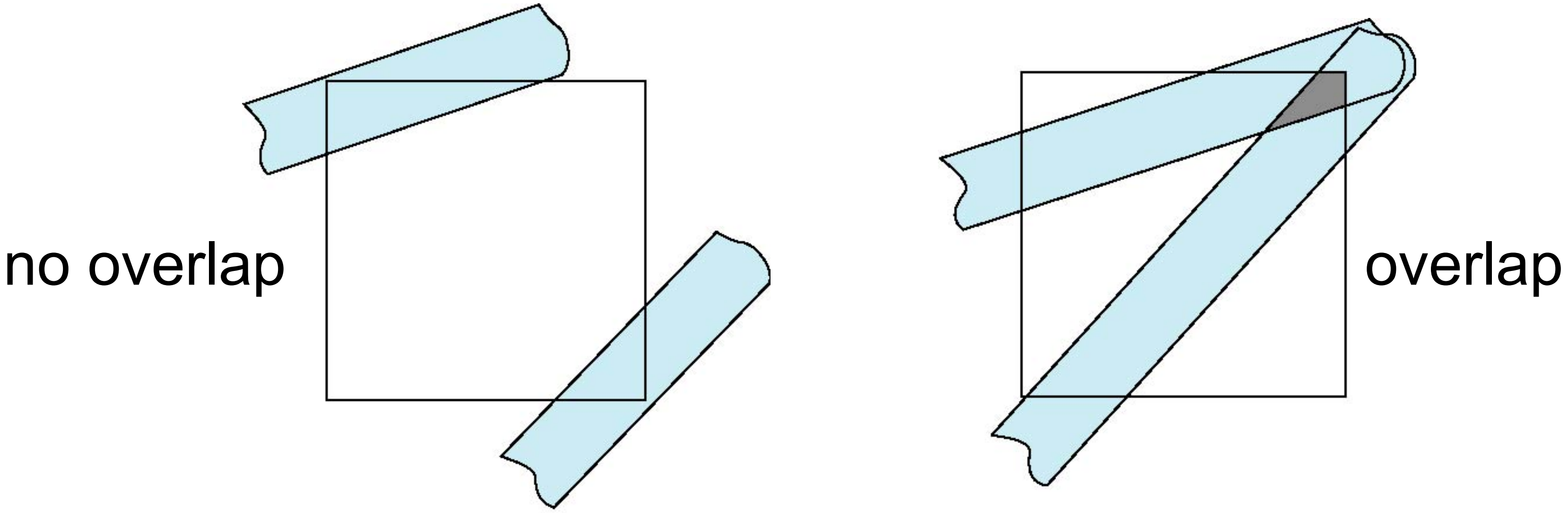
- Ideal raster line is one pixel wide
- All line segments, other than vertical and horizontal segments, partially cover pixels
- Simple algorithms color only whole pixels
- Lead to the “jaggies” or aliasing
- Similar issue for polygons



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

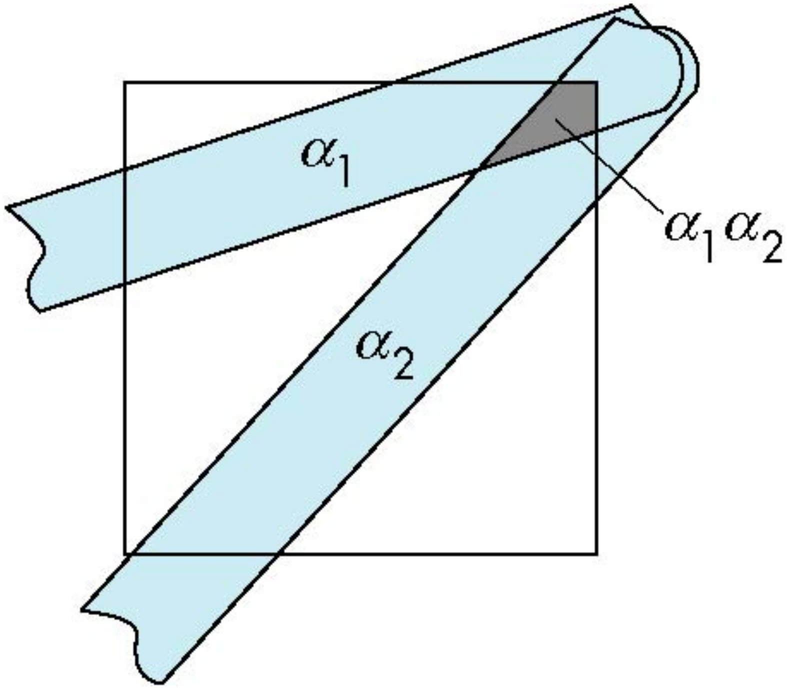
Antialiasing

- Color a pixel by adding in a fraction of the fragment's color
  - Fraction depends on percentage of pixel covered by object
  - Fraction depends on whether there is overlap



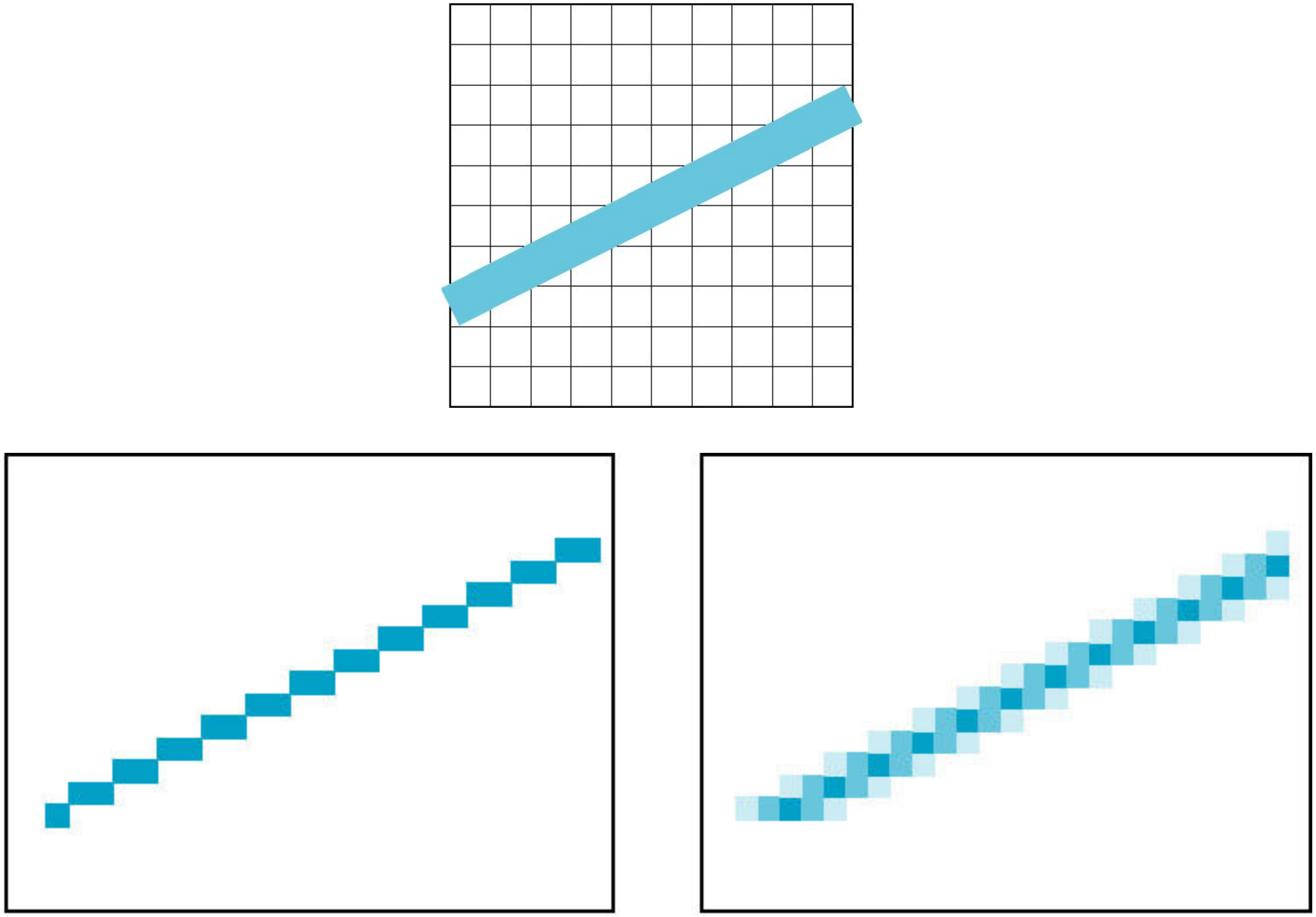
Area Averaging

- Use average area $\alpha_1+\alpha_2-\alpha_1\alpha_2$ as blending factor



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

Area Averaging



OpenGL Antialiasing

- Can enable separately for points, lines, or polygons

```
glEnable(GL_POINT_SMOOTH);
glEnable(GL_LINE_SMOOTH);
glEnable(GL_POLYGON_SMOOTH);
```

- Assigns fractional alpha values along edges
- Based on pixel coverage

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
                    GL_ONE_MINUS_SRC_ALPHA);
```

Multisampling

- If available and enabled(**`GL_MULTISAMPLE`**) multiple *samples* are generated per pixel
- Each *sample* - color, depth and stencil value
- If fragment shader is called for each *sample*, shader must be *sample*-aware
  - **`sample in vec4 color`**
  - **`gl_SamplePosition`**

- All samples are combined to produce the color, depth and stencil value for pixel
- If available, slows performance

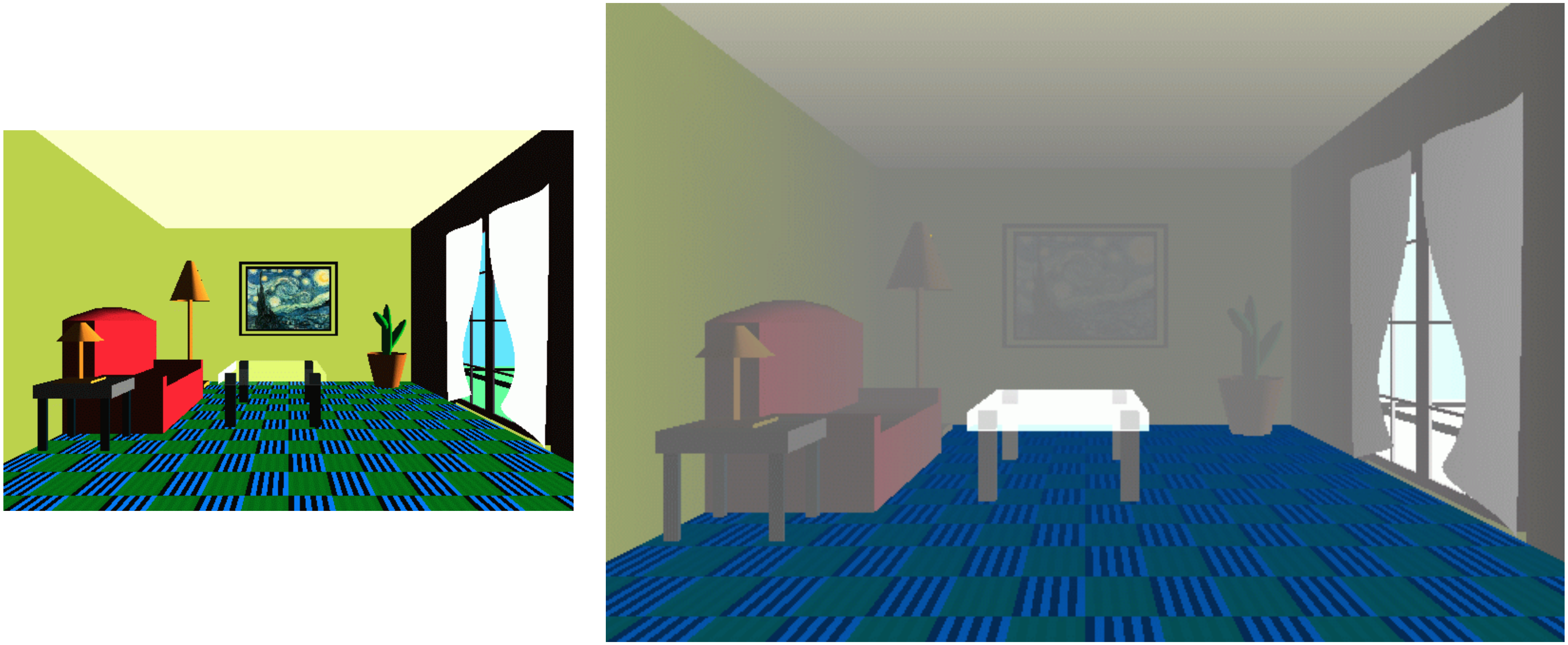
Fog

- We can blend with a fixed color and have the blending factors depend on depth
  - Simulates a fog effect
- Blend source color C_s and fog color C_f by

$$C_s' = f\, C_s + (1-f)\, C_f$$

- f is the *fog factor*
  - Exponential
  - Gaussian
  - Linear (depth cueing)
- Hard-coded fog deprecated but can recreate

Fog Effect



http://www.engin.swarthmore.edu/~jshin1

Picking

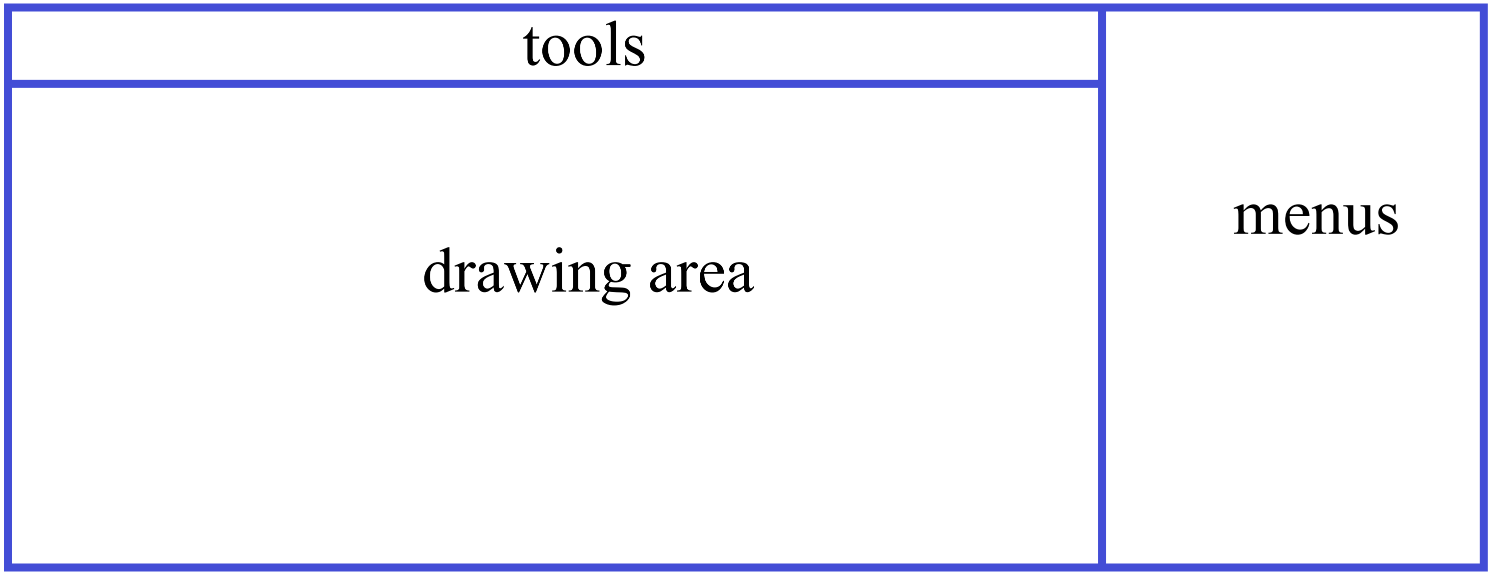
- Identify a user-defined object on the display
- In principle, it should be simple because the mouse gives the position and we should be able to determine to which object(s) a position corresponds
- Practical difficulties
  - Pipeline architecture is feed forward, hard to go from screen back to world
  - Complicated by screen being 2D, world is 3D
  - How close do we have to come to object to say we selected it?

Two Approaches

- Rectangular maps
  - Easy to implement for many applications
  - Divide screen into rectangular regions
- Use back or some other buffer to store object ids as the objects are rendered

Using Regions of the Screen

- Many applications use a simple rectangular arrangement of the screen
  - Example: paint/CAD program

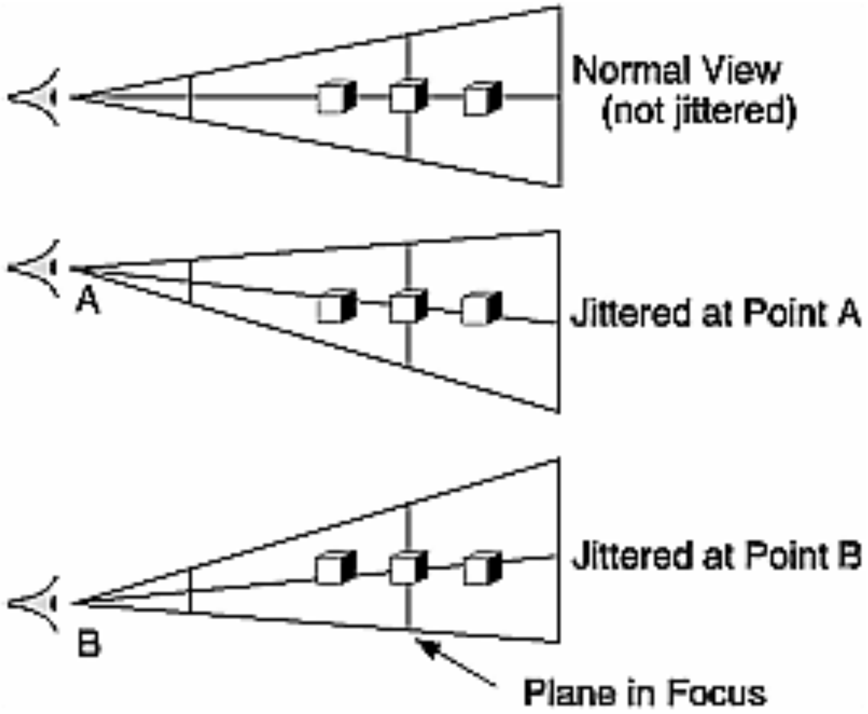


Using another buffer and colors for picking

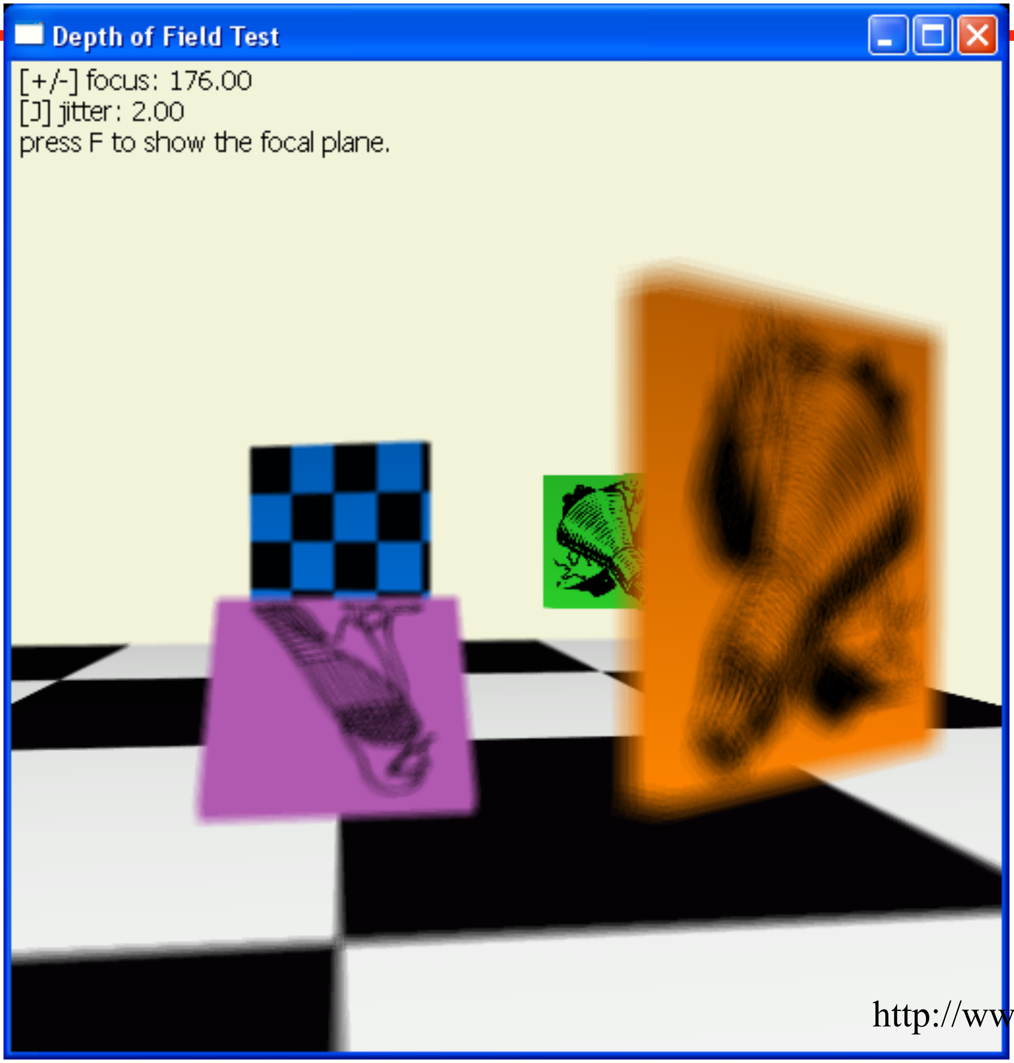
- Can assign a unique color to each object
- Then render the scene to an alternate color buffer (other than the front/back buffer) so the results of the rendering are not visible
- Then get the mouse position and use **`glReadPixels()`** to read the color in the alternate buffer at the position of the mouse
- The returned color gives the id of the picked object

Interactive Depth-of-Field

- Jitter camera
- Each frustum has common plane “in focus”
- Accumulate & blend images



Interactive Depth-of-Field



[+/-] focus: 176.00
[J] jitter: 2.00
press F to show the focal plane.

http://www.cs.stevens.edu/~quynh

Reflections

- One of the most noticeable effect of inter-object lighting
- Direct calculation of the physics (ray tracing) is too expensive
- Our focus is to capture the most significant reflection while minimizing the overhead via rendering the “virtual object”

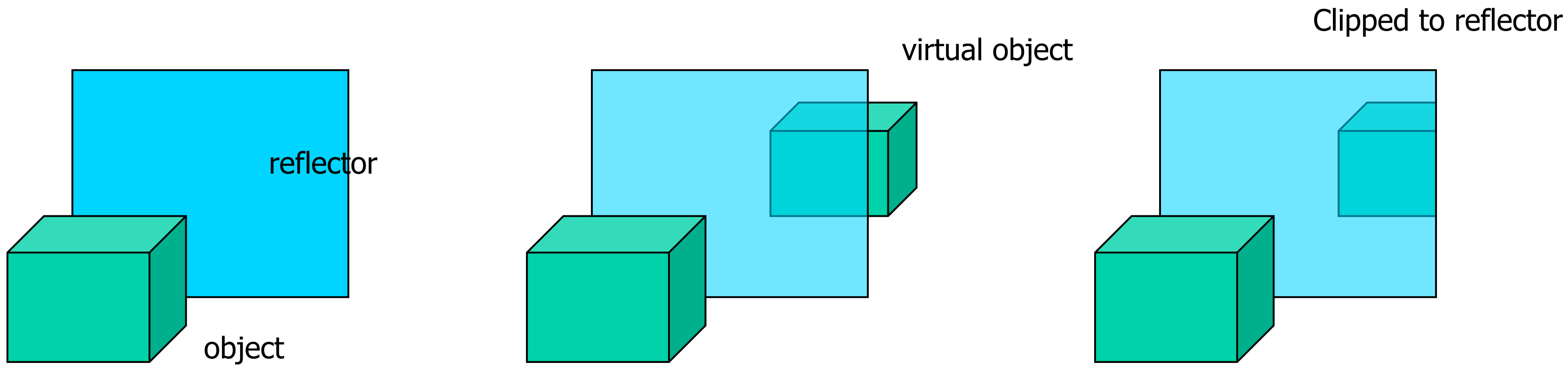


Image vs. Object Space Methods

- Image space methods: create a texture from a view of the reflected objects and apply it to the reflector
    - Advantage: does not depend on the object geometry
    - Disadvantage: sampling issue and also only an approximation (environment mapping as an example)
- Object space methods: create the actual geometry of the object to be reflected and render it to the reflector
    - Disadvantage: accuracy of the geometry
    - Advantage: more accurate reflection (for nearby objects)
- Both methods need to create the virtual objects

Planar Reflections

- The most common reflection – flat mirror, floor, wall, etc
- Creating virtual objects (or reflected objects) is much easier
- A view independent operation – only consider the relative position of the object and the reflector
- The virtual object is created by transforming the object across the reflector plan

Planar Reflections

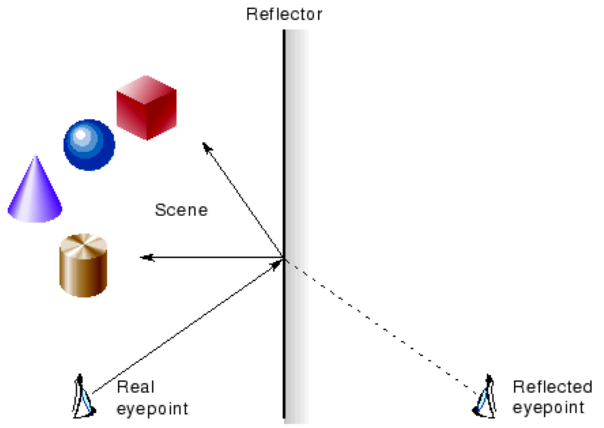


Figure 54. Mirror Reflection of the Viewpoint

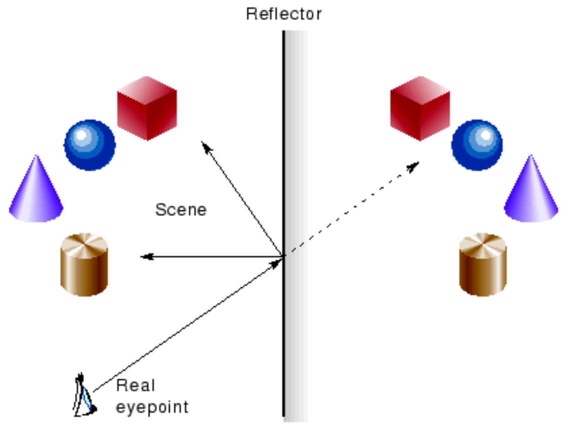
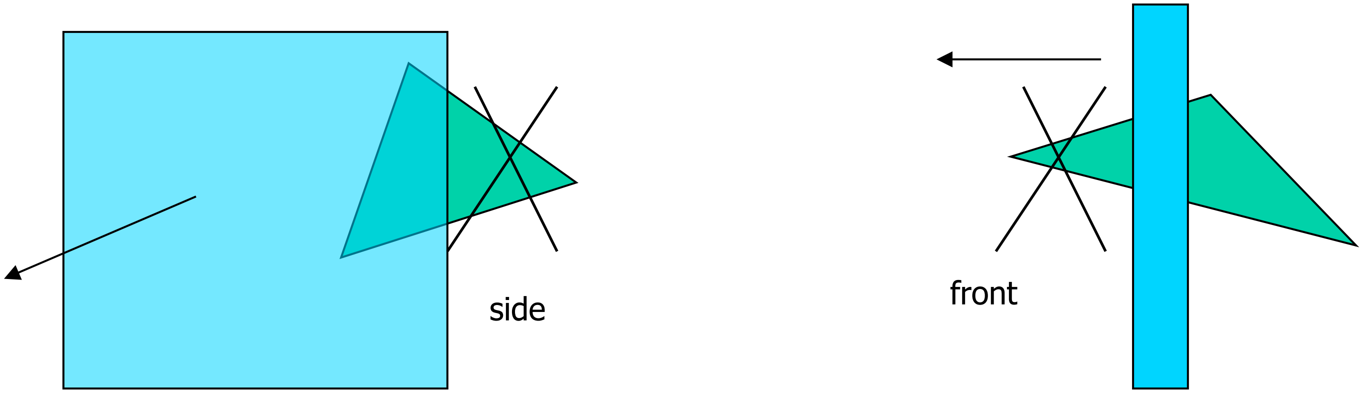


Figure 55. Mirror Reflection of the Scene

Render the Reflected Geometry

- An important task: clip the reflected geometry so it is only visible on the reflector surface
  - Beyond the reflector boundaries and in front of reflector



Angel: Interactive Computer Graphics 3E © Addison-Wesley 2002

Clipping using the stencil

- The key is you only want the reflected geometry to appear on the reflector surface
- Use stencil buffer:
  - Clear the stencil buffer
  - Render the reflector and set the stencil
  - Render the reflected geometry only where the stencil pixels have been set
- The above algorithm uses the stencil buffer to control where to draw the reflection

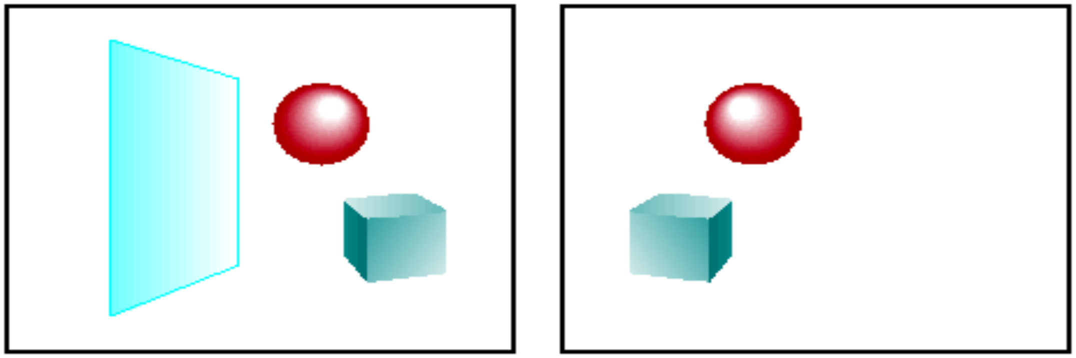
Clipping using the stencil

- Another method: render the reflected object first, and then render the reflector to set the stencil buffer, then clear the color buffer everywhere except where the stencil is set

- This method is to use the stencil buffer to control where to erase the incorrect reflection

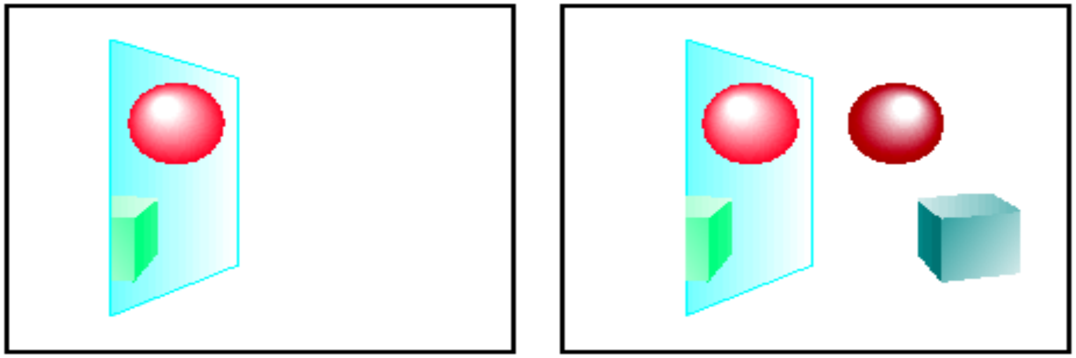
- Advantage: when it is faster to use stencil to control clearing the scene than drawing the entire scene with stencil tests

The stencil erase algorithm



Original scene

Reflected objects

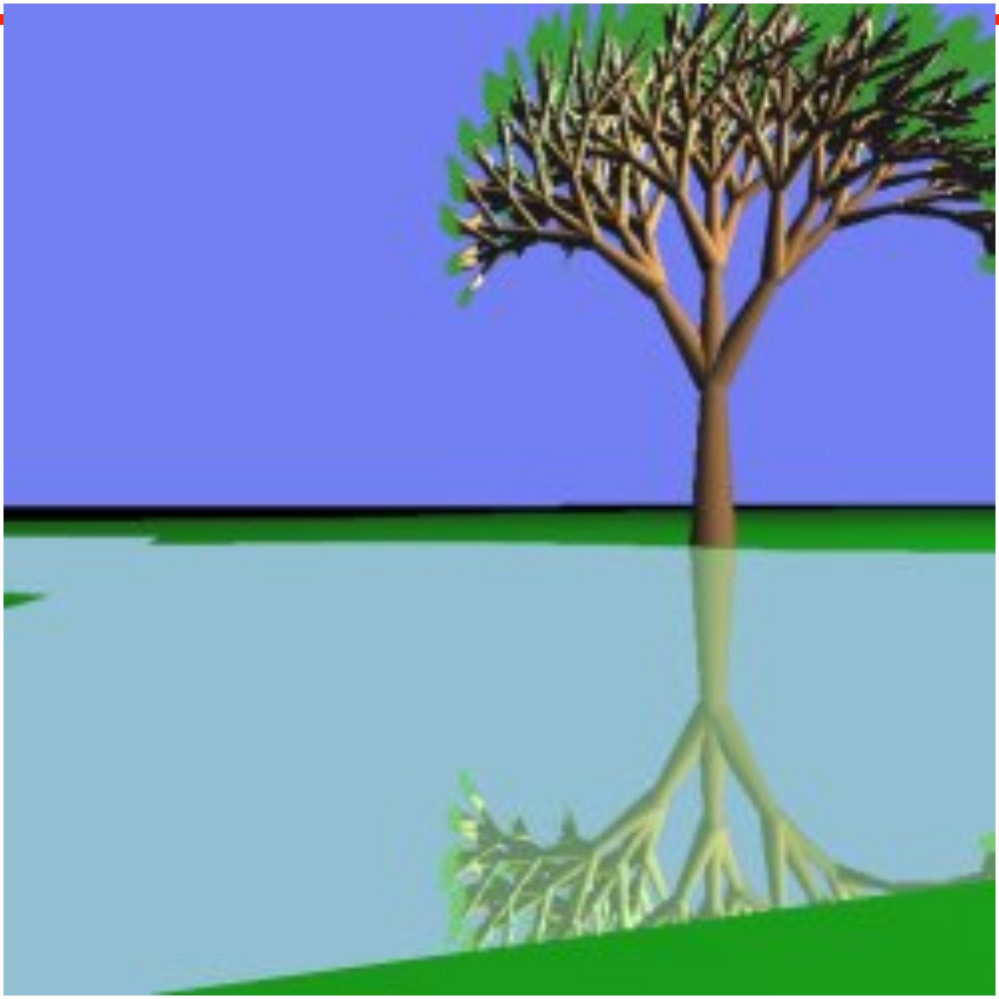


Stencil set outside reflector
and color buffer cleared

Original scene rendered
after reflected scene

Figure 57. Stencil Reflection Steps

Reflection Effect



http://www.cunny.org.uk

Other Applications

- Compositing
- Image Filtering (convolution)
- Motion effects

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012