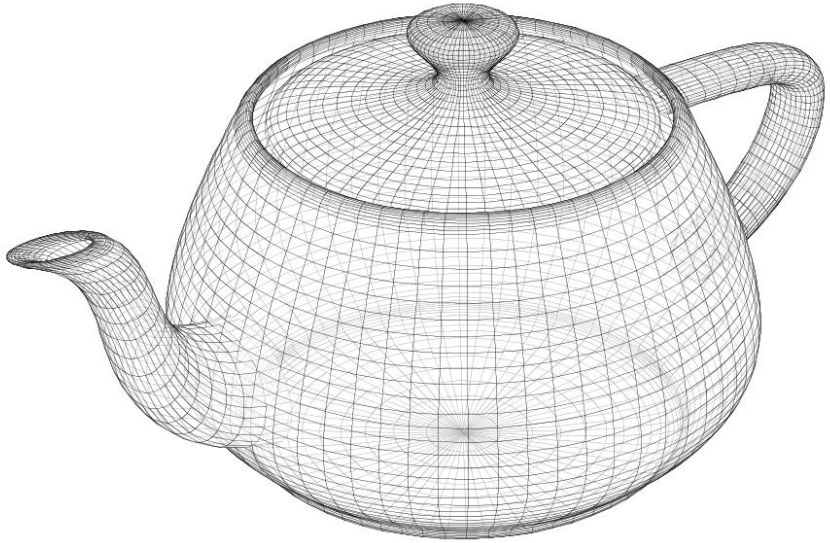


Compositing

Interactive Computer Graphics

Eric Shaffer

ILLINOIS

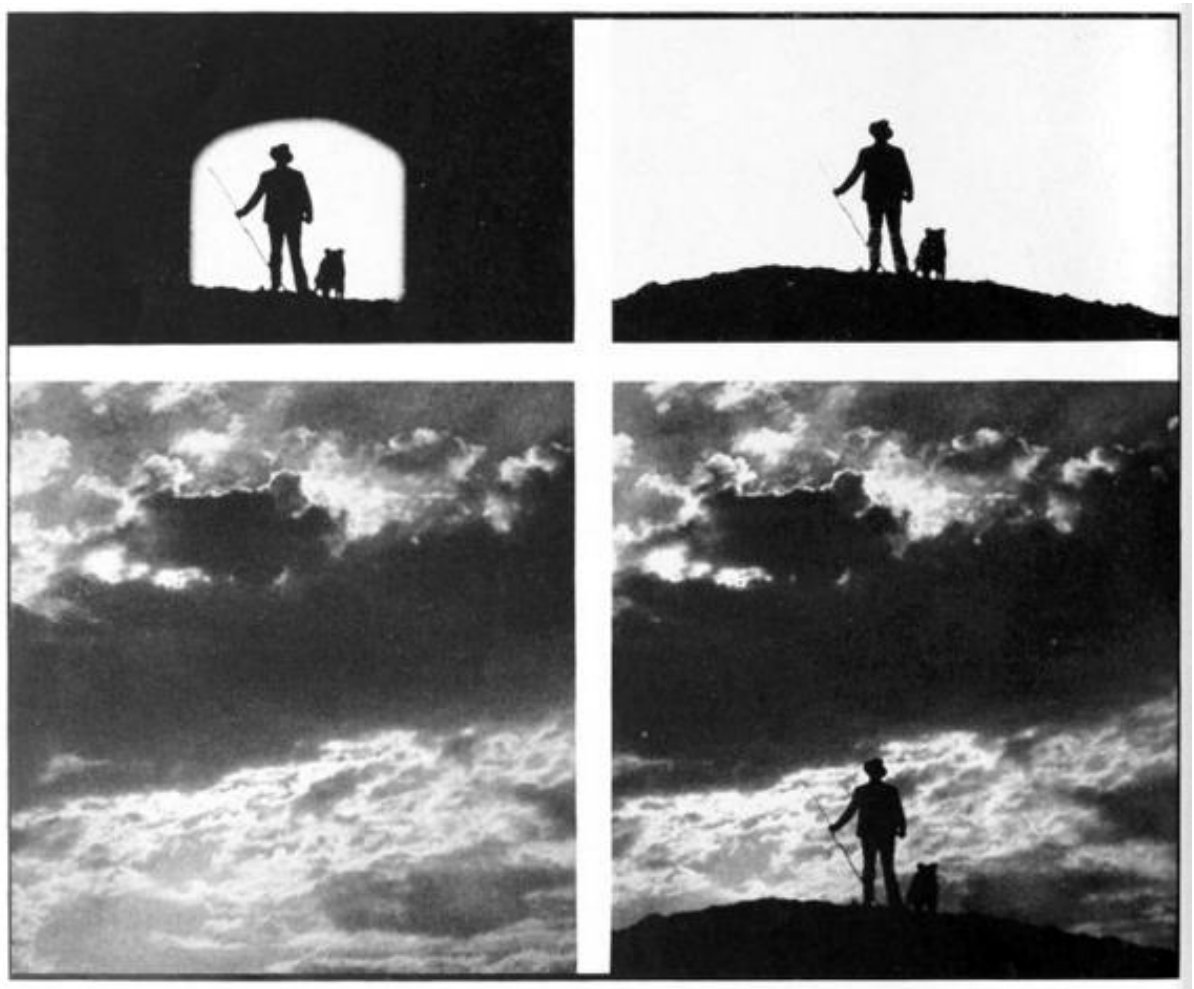
Lynwood Dunn (1904-1998)

Visual effects pioneer

Acme-Dunn optical printer

- Run film through a projector and re-photograph it
- Can zoom in or out, applies filters etc.

Compositing Example



ILLINOIS

Academy of Motion Picture Arts & Sciences

Scientific and Engineering Award

To Alvy Ray Smith, Tom Duff, Ed Catmull and Thomas Porter for their Pioneering Inventions in **Digital Image Compositing**.

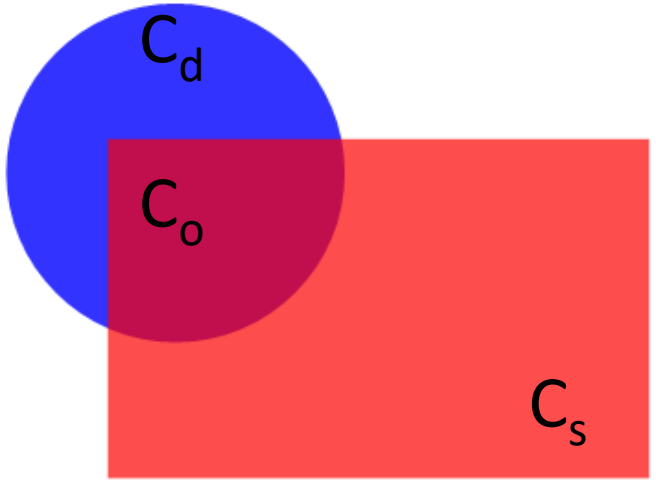
PRESENTED MARCH 2, 1996

Compositing with the Over Operator

$$\mathbf{c}_o = \alpha_s \mathbf{c}_s + (1 - \alpha_s)\mathbf{c}_d \quad [\textbf{over operator}]$$

A color value is given by $c = (r, g, b, \alpha)$

- C_o is the output color
- C_s is the source color
  - this color goes Over destination
- C_d is the destination color
  - the is the pixel color before blending
  - **assumed to be opaque** $(\boldsymbol{\alpha = 1})$

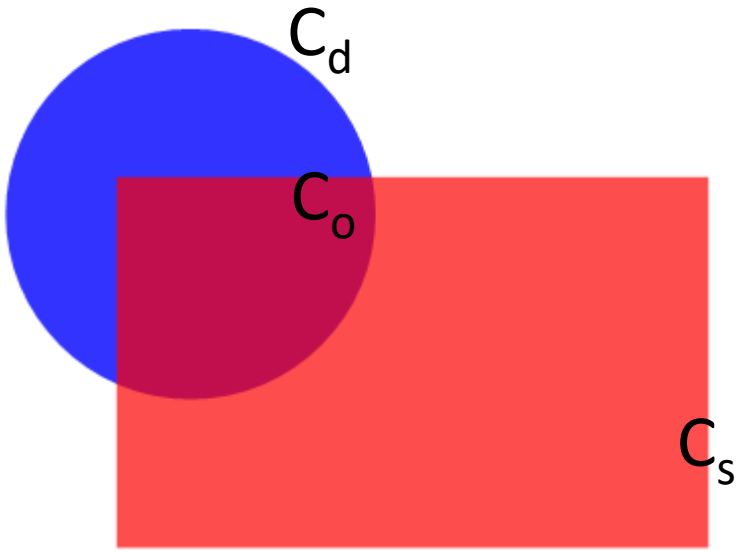


Use the alpha channel of color values when ***alpha blending (compositing)***

- α is in [0,1] with 0 being transparent and 1 being opaque

ILLINOIS

Example



EXAMPLE: BLENDING. A red semitransparent object is rendered onto a blue background. Say that at some pixel the RGB shade of the object is $(0.9, 0.2, 0.1)$, the background is $(0.1, 0.1, 0.9)$, and the object's opacity is set at 0.6. The blend of these two colors is then

$$0.6(0.9, 0.2, 0.1) + (1 - 0.6)(0.1, 0.1, 0.9),$$

which gives a color of $(0.58, 0.16, 0.42)$. □

ILLINOIS

How Well Does It Match Reality?

(Photograph courtesy of Morgan McGuire.)
Real-Time Rendering, Fourth Edition

Alpha blending simulates gauzy fabric well

Less convincing simulating other effects

- Viewing through colored glass or plastic
- A red filter held in front of a blue object in the real world usually makes the blue object look dark.

ILLINOIS

Blending with Over: Requires Back-to-Front Order

To render transparent objects properly:

We need to draw them after the opaque objects

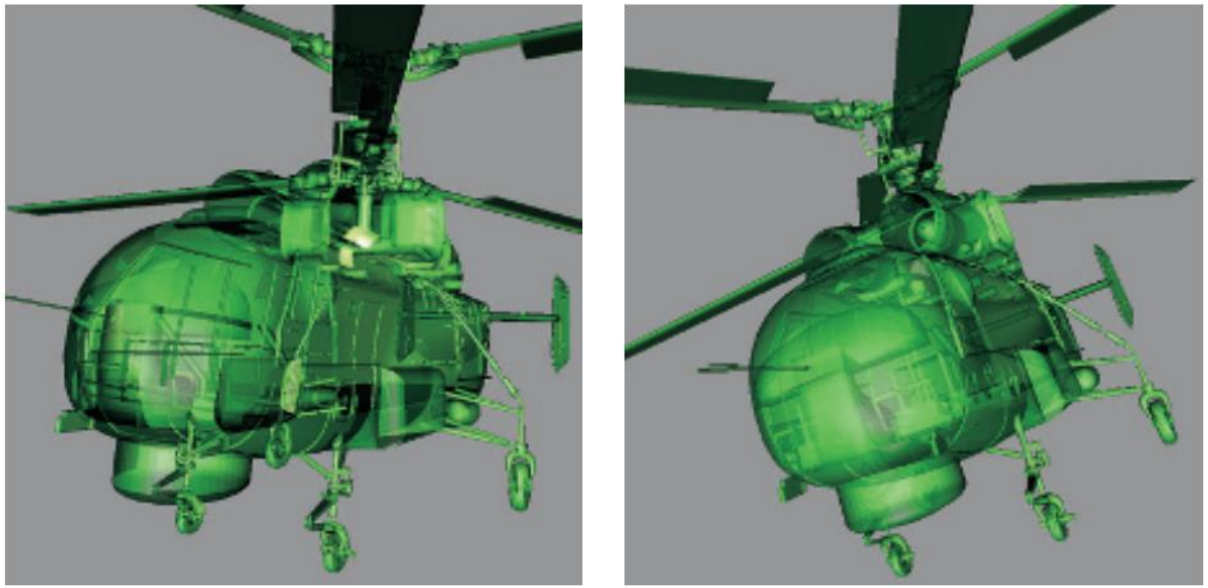
- render all opaque objects first with blending off
- then render transparent objects with over on

Z-buffer (depth buffer) only stores one object per pixel

If several transparent objects overlap the same pixel
z-buffer cannot hold and resolve the effects

Transparent surfaces must be rendered in back-to-front order

Not doing so can give incorrect perceptual cues



On the left the model is rendered with transparency using the z-buffer. Rendering the mesh in an arbitrary order creates serious errors. On the right, depth peeling provides the correct appearance, at the cost of additional passes.

Real-Time Rendering, Fourth Edition (Page 152)

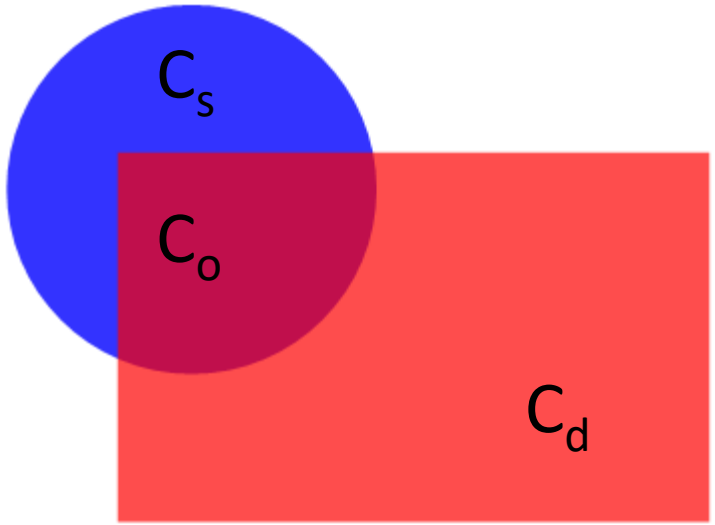
To render back-to-front, you can sort objects by their centroids along the view direction
How does this method sometimes fail?

ILLINOIS

Blending Front-to-Back: Under Operator

$$\mathbf{c}_o = \alpha_d \mathbf{c}_d + (1 - \alpha_d)\alpha_s \mathbf{c}_s \quad \lfloor \textbf{under operator} \rfloor,$$
$$\mathbf{a}_o = \alpha_s(1 - \alpha_d) + \alpha_d = \alpha_s - \alpha_s\alpha_d + \alpha_d.$$

- Here the source and destination designations are swapped
- We need to compute and carry along α values



Why was alpha not computed for the over operator?

ILLINOIS

Pre-multiplied Alpha

With pre-multiplied alpha, a color value is given by $c = (\alpha r, \alpha g, \alpha b, \alpha)$

- Sometimes called *associated alpha*
- *Unassociated alpha* = non-pre-multiplied alpha

- Can blend using the over operator and pre-multiplied alpha

$$\mathbf{c}_o = \mathbf{c}'_s + (1 - \alpha_s)\mathbf{c}_d$$

- Here $c' = (\alpha r, \alpha g, \alpha b)$
- Again, c_d is assumed to be opaque

ILLINOIS

Should You Use Pre-multiplied Alpha?

Yes - WebGL sort of expects you to use pre-multiplied alpha when blending

Over operator using pre-multiplied alpha is provably associative
…using non-pre-multiplied alpha…maybe not?

- It is best to use pre-multiplied data whenever filtering and blending is performed, as operations such as linear interpolation do not work correctly using unmultiplied alphas

- Pre-multiplied alphas also allow cleaner theoretical treatment

- ***Real-Time Rendering***, Fourth Edition

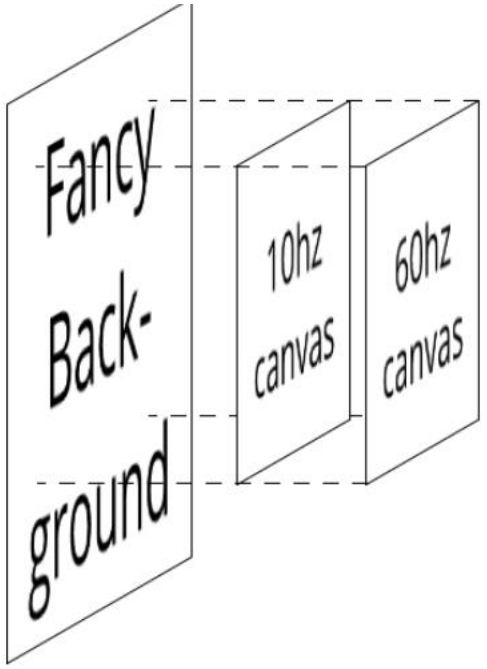
ILLINOIS

WebGL Canvases are Composited

A browser composites your WebGL canvas on the displayed page

This is done using pre-multipled alpha

You could stack multiple canvasses



If yes, you do want to blend with the webpage then do one of the following

- Make sure the values you write into the canvas are premultiplied alpha values.
- Tell the browser the values in the canvas are not premultiplied

```
var gl = someCanvas.getContext("webgl", {premultipliedAlpha: false});
```

On top of that by default, images loaded into WebGL use *un-premultiplied alpha* Which means you either need to

- Set your canvas to not be premultiplied
- Do the multiplication yourself in your shader

```
gl_FragColor.rgb *= gl_FragColor.a;
```

- Tell WebGL to premultiply the texture when you load it into WebGL

```
gl.pixelStorei(gl.UNPACK_PREMULTIPLY_ALPHA_WEBGL, true);
```

Assuming your canvas is premultiplied you want your blend function to be

```
gl.blendFunc(gl.ONE, gl.ONE_MINUS_SRC_ALPHA);
```

https://stackoverflow.com/questions/39341564/webgl-how-to-correctly-blend-alpha-channel-png

ILLINOIS

You Can Disable Canvas Compositing

Do you or do you not want your WebGL image blended with the webpage?

If no, you don't want to blend with the webpage then do one of the following

- Turn off alpha in the canvas

```
var gl = someCanvas.getContext("webgl", { alpha: false });
```

- Make sure your alpha stays at 1.0

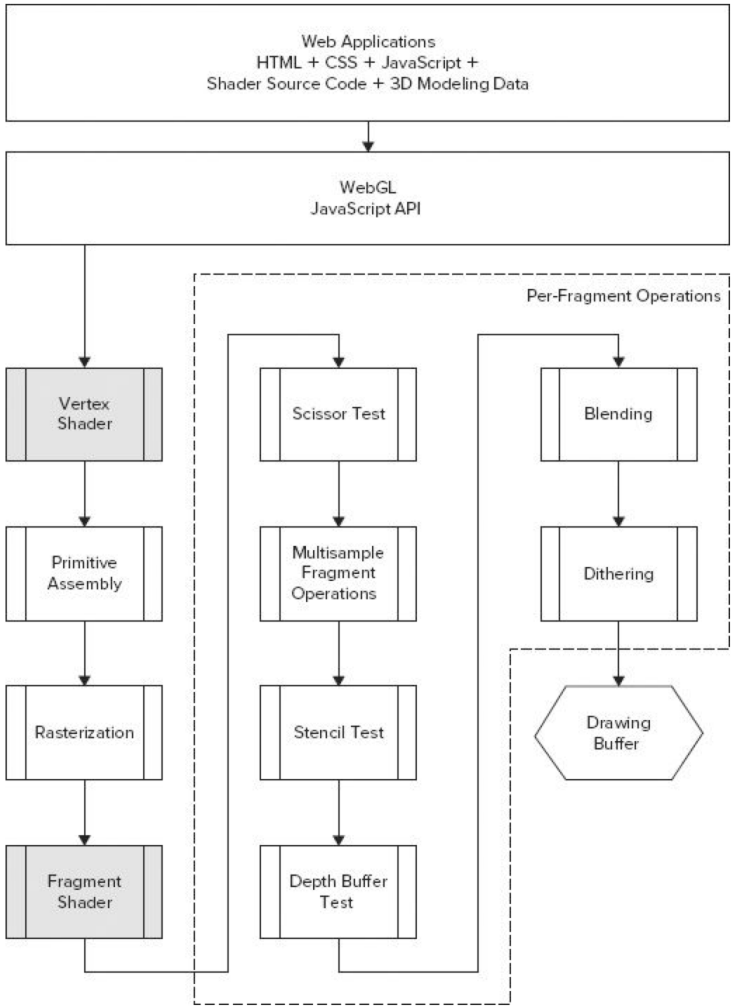
  The easy way to do that is to just clear it after rendering with

```
gl.colorMask(false, false, false, true);
gl.clearColor(0, 0, 0, 1);
gl.clear(gl.COLOR_BUFFER_BIT);
```

https://stackoverflow.com/questions/39341564/webgl-how-to-correctly-blend-alpha-channel-png

ILLINOIS

WebGL Pipeline



Scissor Test:
cull pixels outside of a rectangular area

Multisample:
anti-aliasing operation

Stencil Test:
uses a stencil buffer to mask pixels
can be used in shadow generation

Depth Buffer Test:
hidden surface removal

Blending:
compositing using alpha channel

ILLINOIS

Blending

```
//enable blending
gl.Enable(gl.BLEND)


//to set up the parameters of the generic blending equation
//call ONE of the functions below
gl.blendFunc(GLenum sfactor, GLenum dfactor);
//OR
gl.blendFuncSeparate(GLenum srcRGB, GLenum dstRGB,
                     GLenum srcAlpha,
                    GLenum dstAlpha);
```

WebGL lets you specify the factors and operations in the generic blending equation:

$$color_{final} = factor_{source} \times color_{source} \ \mathbf{op} \ factor_{dest} \times color_{dest}$$

ILLINOIS

Blending

$color_{final} = factor_{source} \times color_{source}$ **op** $factor_{dest} \times color_{dest}$

FUNCTION	RGB BLEND FACTORS	ALPHA BLEND FACTOR
gl.ZERO	(0, 0, 0)	0
gl.ONE	(1, 1, 1)	1
gl.SRC_COLOR	(R_s, G_s, B_s)	A_s
gl.ONE_MINUS_SRC_COLOR	$(1, 1, 1) - (R_s, G_s, B_s)$	$1 - A_s$
gl.DST_COLOR	(R_d, G_d, B_d)	A_d
gl.ONE_MINUS_DST_COLOR	$(1, 1, 1) - (R_d, G_d, B_d)$	$1 - A_d$
gl.SRC_ALPHA	(A_s, A_s, A_s)	A_s
gl.ONE_MINUS_SRC_ALPHA	$(1, 1, 1) - (A_s, A_s, A_s)$	$1 - A_s$
gl.DST_ALPHA	(A_d, A_d, A_d)	Ad
gl.ONE_MINUS_DST_ALPHA	$(1, 1, 1) - (A_d, A_d, A_d)$	$1 - A_d$
gl.CONSTANT_COLOR	(R_c, G_c, B_c)	A_c
gl.ONE_MINUS_CONSTANT_COLOR	$(1, 1, 1) - (R_c, G_c, B_c)$	$1 - A_c$
gl.CONSTANT_ALPHA	(A_c, A_c, A_c)	A_c
gl.ONE_MINUS_CONSTANT_ALPHA	$(1, 1, 1) - (A_c, A_c, A_c)$	$1 - A_c$
gl.SRC_ALPHA_SATURATE	(f, f, f)	1

ILLINOIS

Changing the Blending Operator

```
gl.blendEquation(Glenum mode);
```

Lets you specify the blending operation.
Addition is the default.

```
//colorfinal = factorsource × colorsource + factordest × colordest
gl.blendEquation(GL_FUNC_ADD);

//colorfinal = factorsource × colorsource – factordest × colordest
gl.blendEquation(GL_FUNC_SUBTRACT);

//colorfinal = factordest × colordest – factorsource × colorsource
gl.blendEquation(GL_FUNC_REVERSE_SUBTRACT);
```

Blending

```
gl.blendFunc(GLenum sfactor, GLenum dfactor);
```

Lets you specify the blending function for both the RGB and Alpha values for both the source and destination

```
gl.blendFunc(gl.SRC_ALPHA,  gl.ONE_MINUS_SRC_ALPHA);
```

Is the traditional way to implement alpha blending in the style if non-pre-multiplied alpha.

Is this correct?

Blending

```
gl.blendFunc(GLenum sfactor, GLenum dfactor);
```

Lets you specify the blending function for both the RGB and Alpha values for both the source and destination

```
gl.blendFunc(gl.SRC_ALPHA,  gl.ONE_MINUS_SRC_ALPHA);
```

Is the traditional way to implement alpha blending in the style if non-pre-multiplied alpha.

Is this correct?

New alpha is calculated incorrectly….should use

```
gl.blendFuncSeparate(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA,
                     gl.ONE, gl.ONE_MINUS_SRC_ALPHA);
```

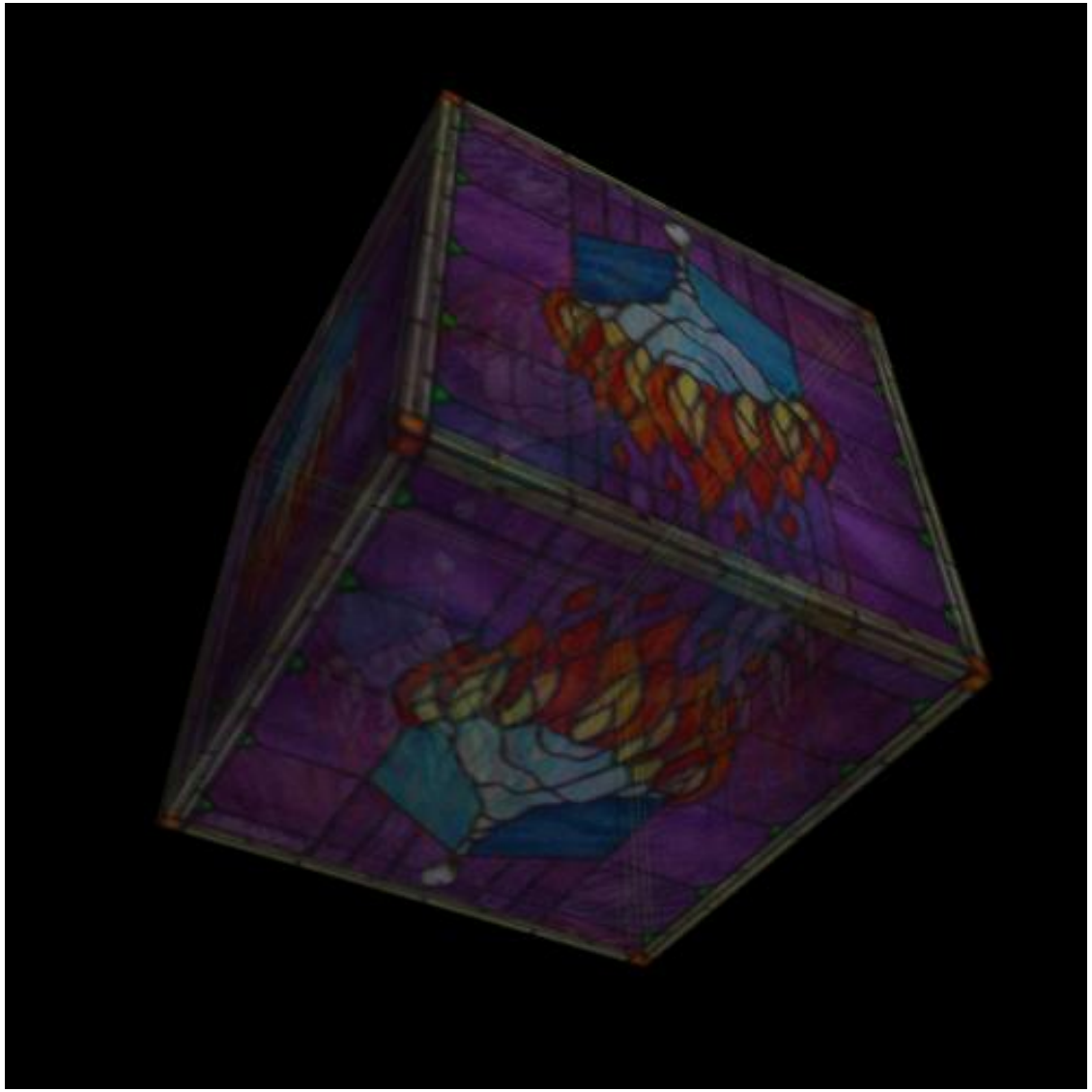
ILLINOIS

Blending and Drawing Order

```
// 1. Enable depth testing, make sure the depth buffer is writable
//    and disable blending before you draw your opaque objects.
gl.enable(gl.DEPTH_TEST);
gl.depthMask(true);
gl.disable(gl.BLEND);

// 2. Draw your opaque objects in any order (preferably sorted on state)
// 3. Keep depth testing enabled, but make depth buffer read-only
//    and enable blending
gl.depthMask(false);
gl.enable(gl.BLEND);

// 4. Draw your semi-transparent objects back-to-front
// 5. If you have UI that you want to draw on top of your
//    regular scene, you can finally disable depth testing
gl.disable(gl.DEPTH_TEST);

// 6. Draw any UI you want to be on top of everything else
```

Anyuru, Andreas
Professional WebGL Programming: Developing 3D Graphics for the Web

Blending in WebGL



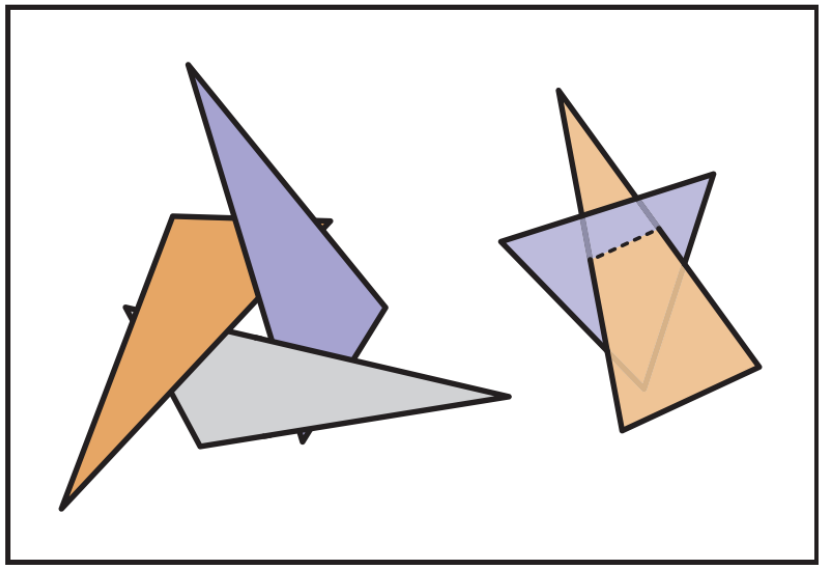
ILLINOIS

Hidden Surface Removal

- Hidden Surface Removal
  - …don’t render surfaces occluded by surfaces in front of them
- Was a significant area of research in early days of CG
  - …lots of algorithms suggested
- Painter’s Algorithm
  - Render objects in order from back to front
    - i.e. sort your triangles by depth and render deepest first
  - Can anyone imagine any problems with this approach?

Problems with the Painter's Algorithm

- No correct rendering order for
  - intersecting triangle
  - occlusion cycles



- Sorting is slow…too slow for interactivity in complex scenes

Hidden Surface Removal: Z-Buffer

Key Observation:
Each pixel displays color of only one triangle,
Ignores everything behind it

Don't need to sort triangles, just find, for each pixel, the closest triangle

Z-buffer: one fixed or floating point value per pixel

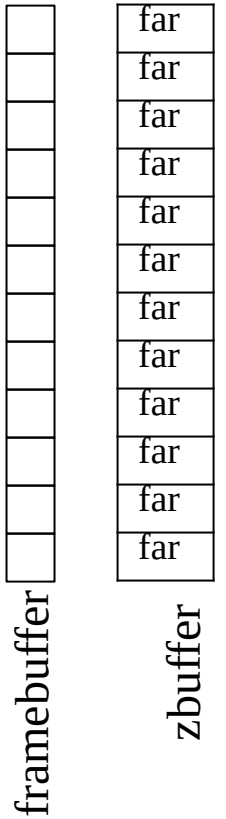
Algorithm:

For each rasterized fragment (x,y)

If z < zbuffer(x,y) then

framebuffer(x,y) = fragment color

zbuffer(x,y) = z



Frame Buffer: buffer that stores the colors for the pixels we will render

ILLINOIS

Z-Buffer

Key Observation:
Each pixel displays color of only one triangle, ignores everything behind it

Don't need to sort triangles,
just find for each pixel the closest triangle

Z-buffer: one fixed or floating point value per pixel

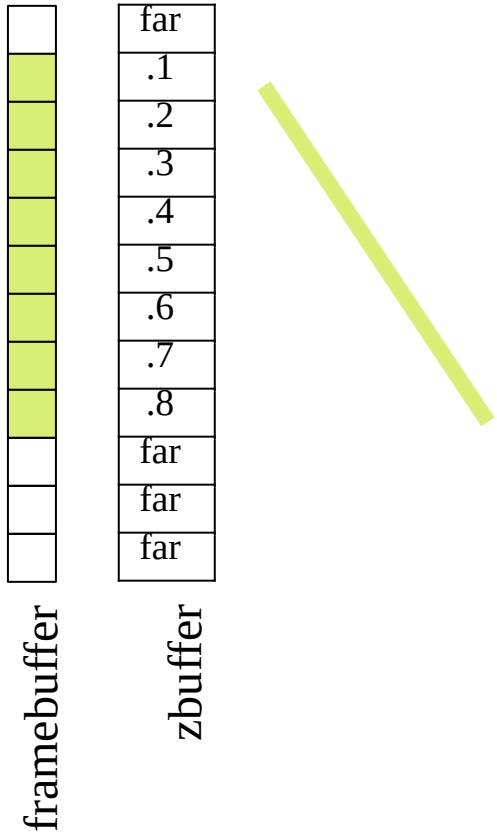
Algorithm:

For each rasterized fragment (x,y)

If z < zbuffer(x,y) then

framebuffer(x,y) = fragment color

zbuffer(x,y) = z



ILLINOIS

Z-Buffer

Key Observation:
Each pixel displays color of only one triangle, ignores everything behind it

Don't need to sort triangles,
just find for each pixel the closest triangle

Z-buffer: one fixed or floating point value per pixel

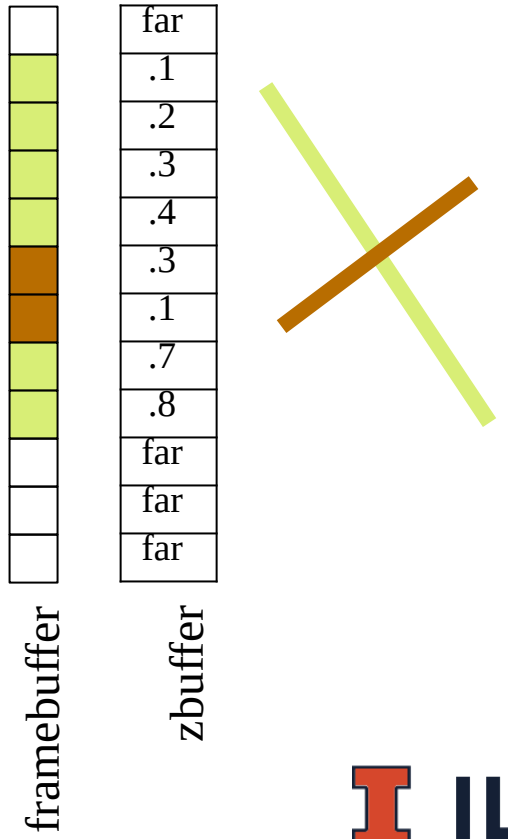
Algorithm:

For each rasterized fragment (x,y)

If z < zbuffer(x,y) then

framebuffer(x,y) = fragment color

zbuffer(x,y) = z



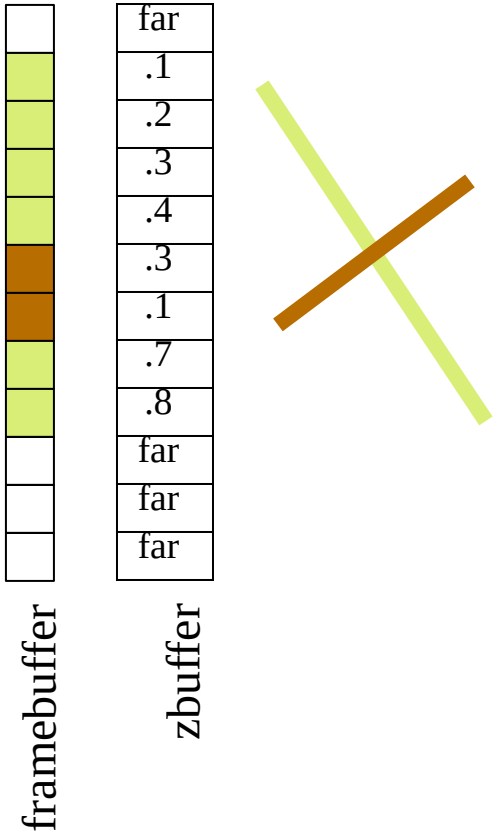
ILLINOIS

Z-Buffer

Get fragment z-values by interpolating z-values at vertices during rasterization

True perspective projection destroys z-values, setting them all to *–d*

The perspective distortion we use preserves at least the ordering of z-values

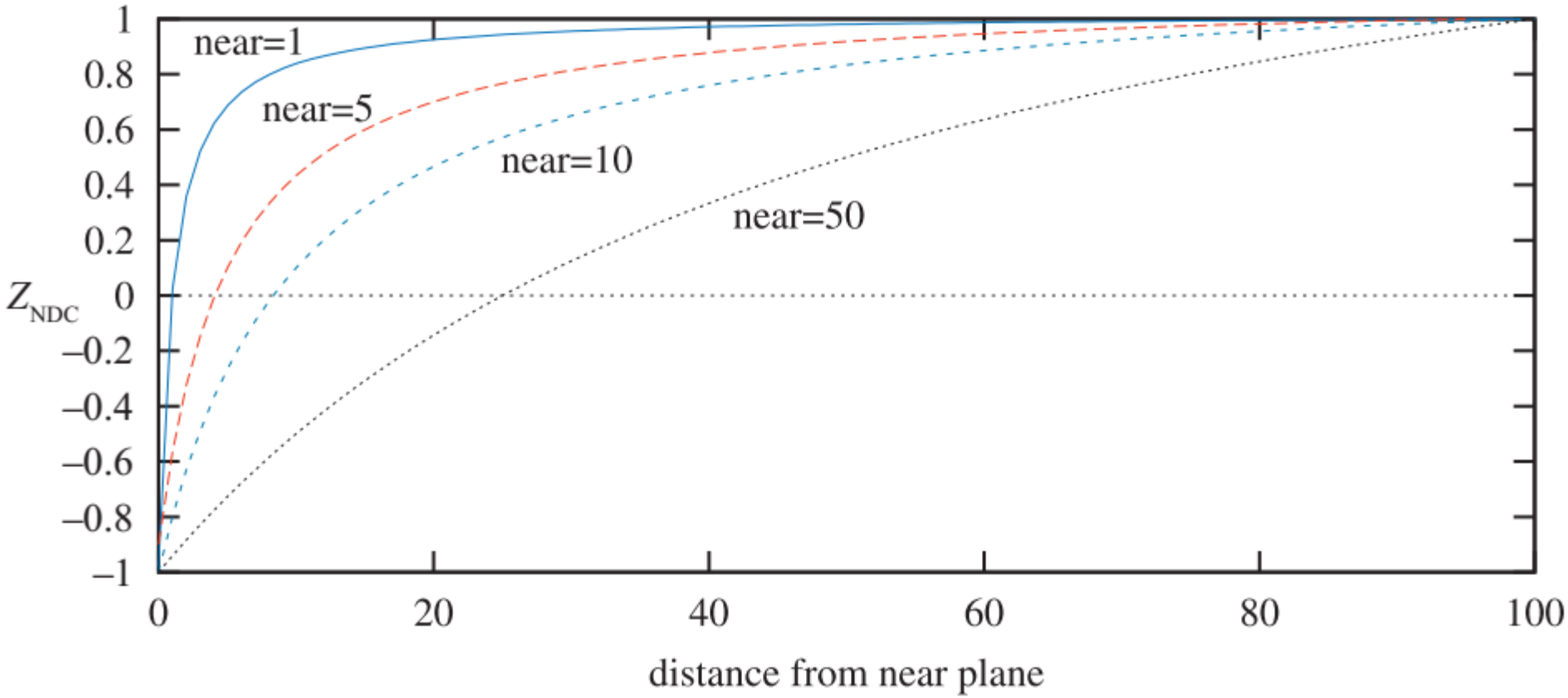


ILLINOIS

Precision Issues with Z-Buffering

- In practice, depths values are typically converted to non-negative integers
  - Comparison operation needs to be fast…
- Imagine having depth values of {0,1,…,B-1}
  - 0 → near clipping plane distance
  - B-1 → far clipping plane distance
- Depths occur discretely in “buckets”
  - Each bucket covers a range of length $\Delta z = \frac{f-n}{B}$
- If we use b bits for the z-buffer values, B = 2^b
  - You usually can’t change the value b
  - To maximize z-buffer effectiveness, **need to minimize f-n**

…More Precision Issues with Z-Buffering

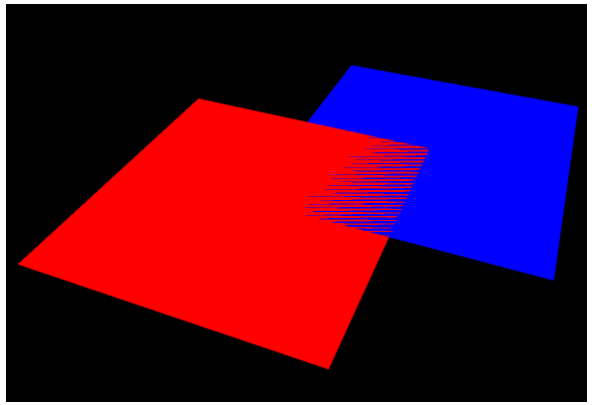


The effect of varying the distance of the near plane from the origin. The distance f−n is kept constant at 100. As the near plane becomes closer to the origin, points nearer the far plane use a smaller range of the normalized device coordinate (NDC) depth space. This has the effect of making the z-buffer less accurate at greater distances.

Real-Time Rendering, Fourth Edition

ILLINOIS

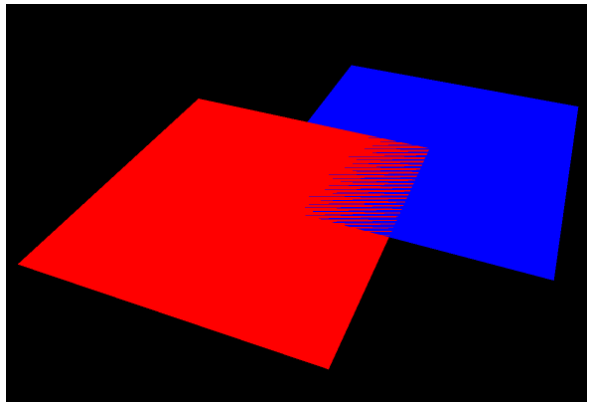
Z-Fighting



How can you fix z-fighting?

ILLINOIS

Z-Fighting



How can you fix z-fighting?

1. Move co-planar polygons slightly away from each other
2. Move near and far clipping planes as close together as you feasibly can

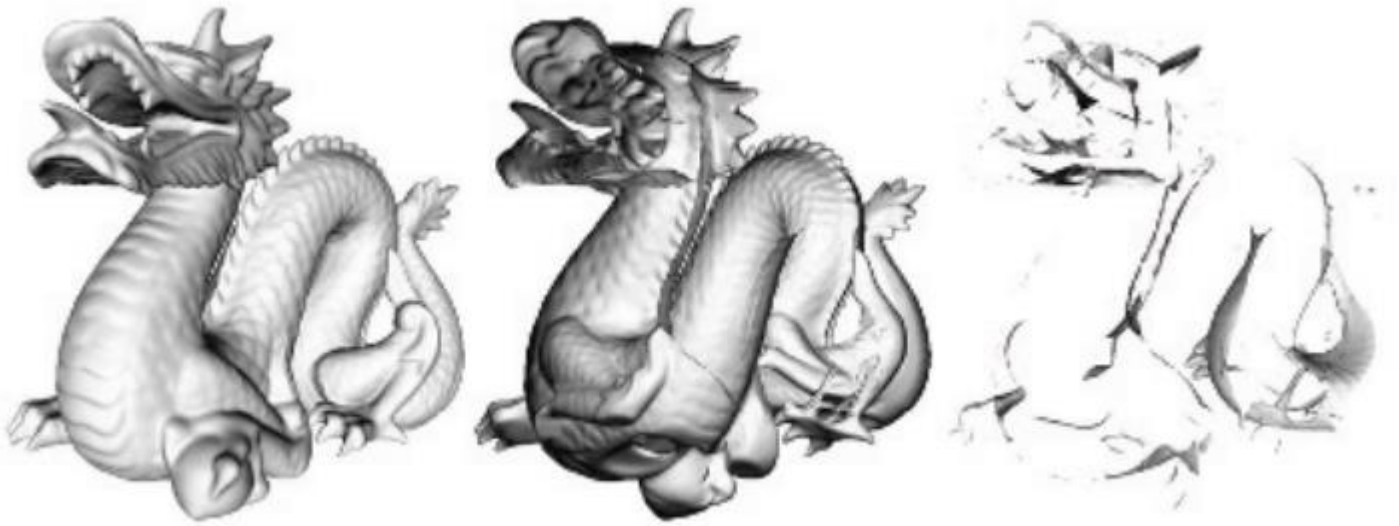
WebGL Hidden Surface Removal

```
gl.enable(gl.DEPTH_TEST);    // use depth test for hidden surface remove
gl.depthFunc(gl.LESS);       //this is the default
gl.clear(gl.DEPTH_BUFFER_BIT); // clear depth values form previous frame
```

Hidden surface removal uses the depth buffer (z-buffer)

Happens after the fragment shader

Order Independent Transparency



Alpha blending works for sorted rendering

- Front to back
- Back to front

Doesn't work for out-of-order

- Front, back, middle

Depth-peeling is an algorithm for order-independent blending

ILLINOIS

Depth Peeling

Cass Everett, NVIDIA

Needs 2 z-buffers (previous, current)

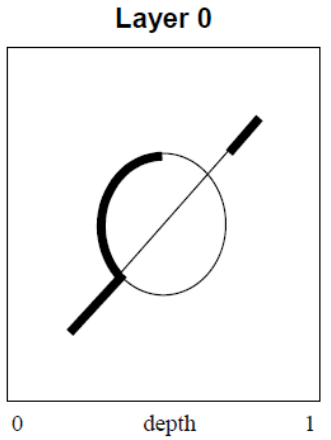
One rendering pass per layer

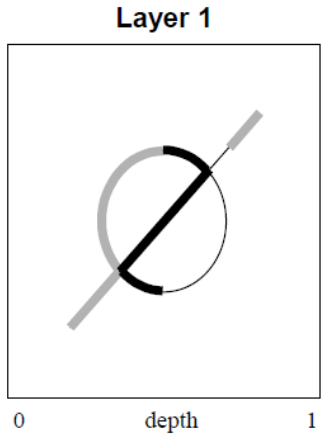
Fragment written to frame buffer if

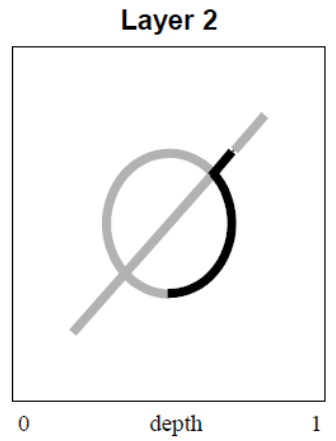
- Farther than previous z-buffer
- Closer than current z-buffer

After each pass, current z-buffer written to previous z-buffer

Surviving fragment composited "under" displayed fragment

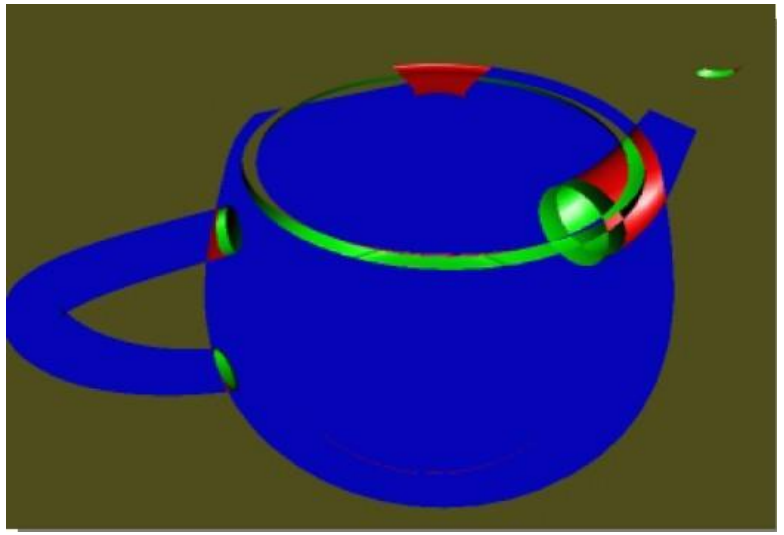


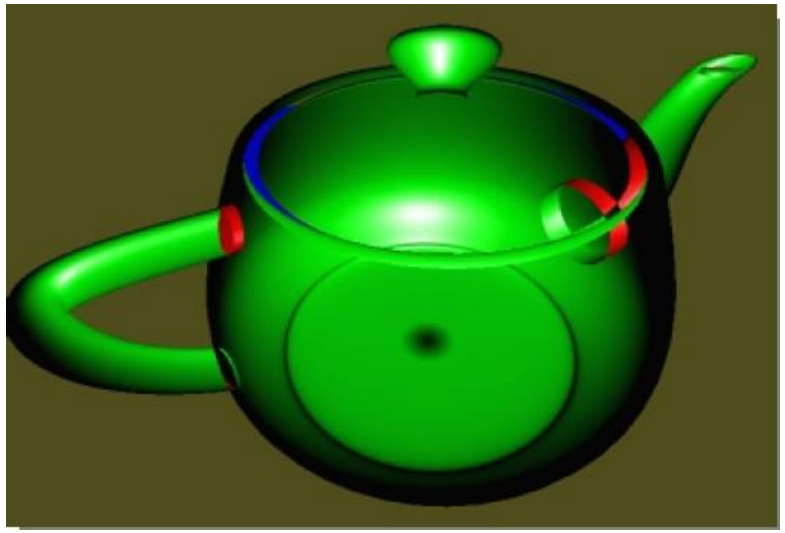


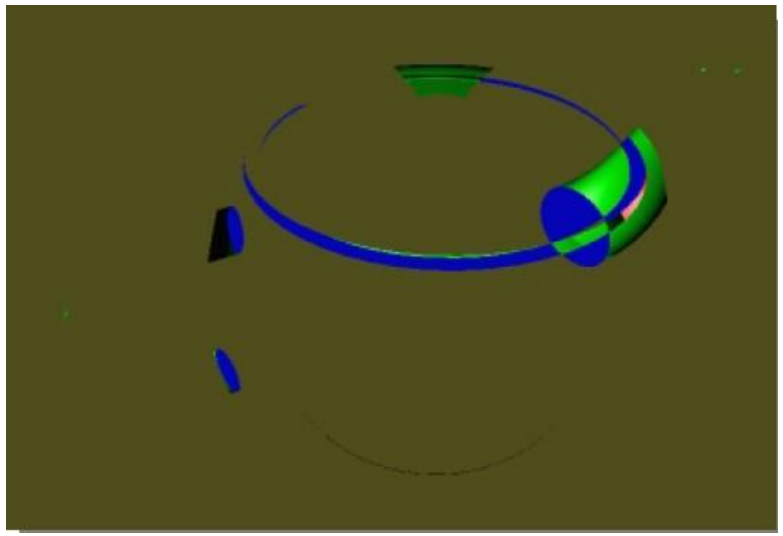


ILLINOIS

Depth Peels – Which Layer is Which?

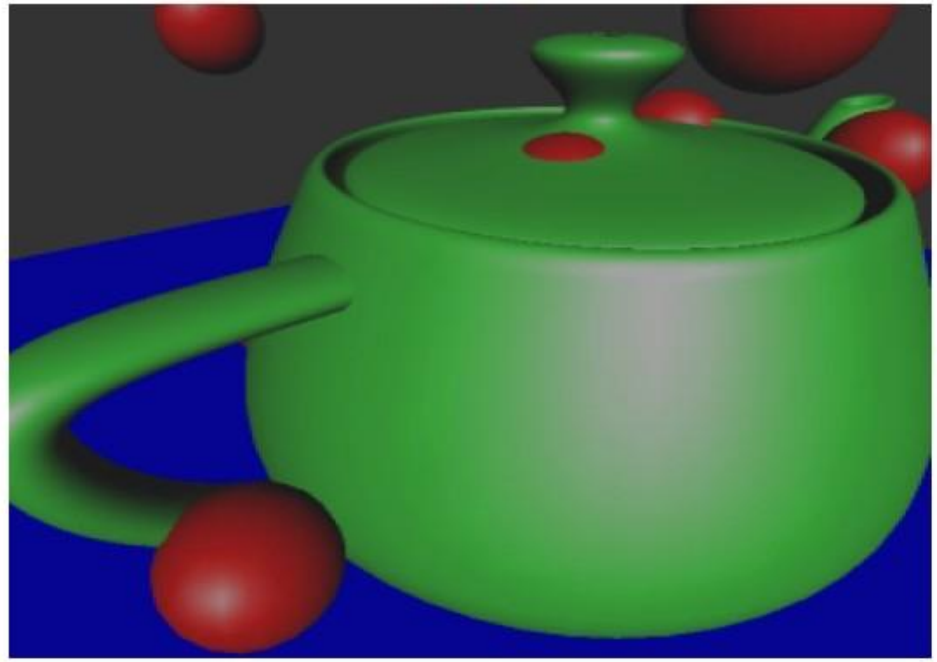




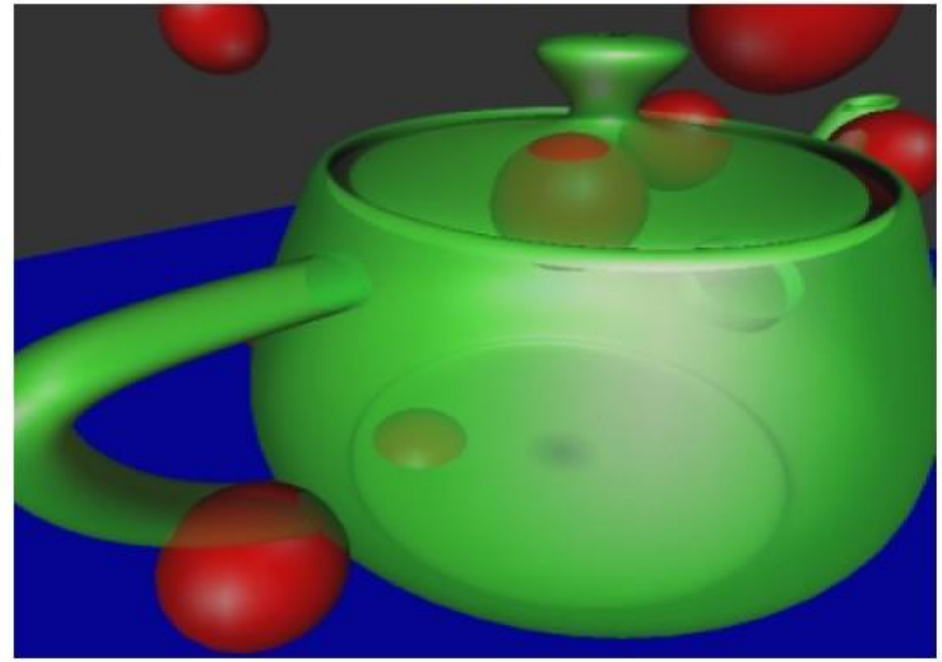


ILLINOIS

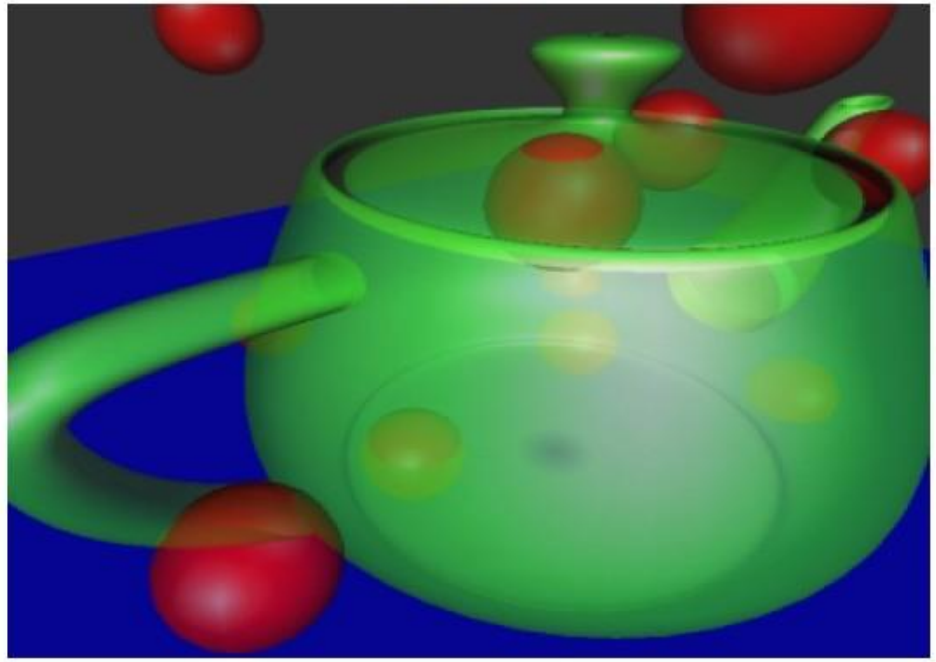
1 layer



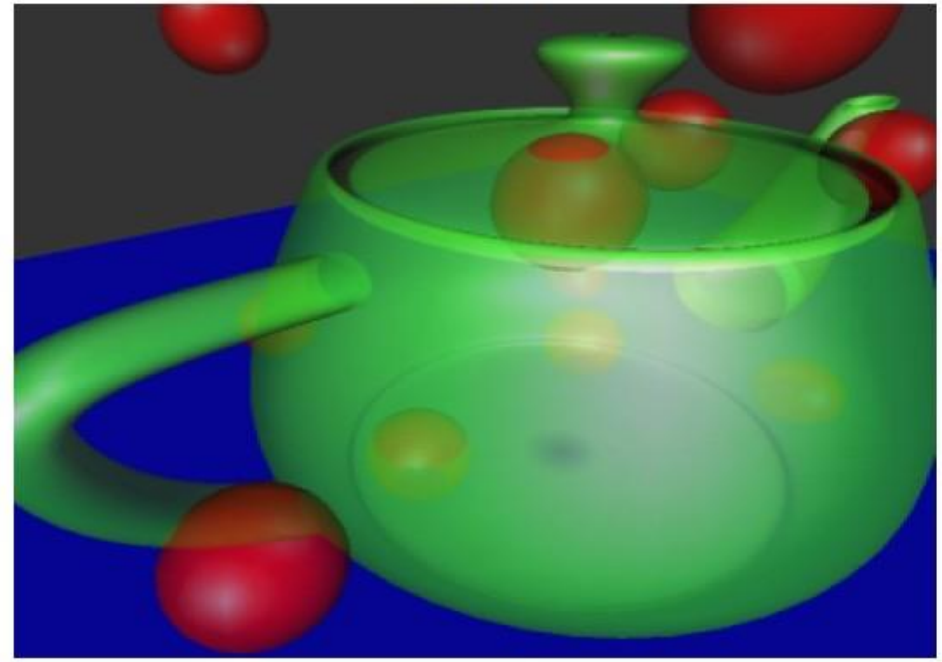
2 layers



3 layers



4 layers



Alternatives to Depth Peeling

- Loren Carpenter proposed the ***A-Buffer*** in 1984
  - Linked list of fragments with depth and alphas at each screen location
  - Problematic for earlier generations of GPUs
  - Even now, memory must be pre-allocated and can easily run out
  - Not commonly used…performance
- ***Multi-layer alpha blending***
  - Possibly sort and blend first few layers correctly
  - Deeper layers blended using two buffers…one for opaque, one transparent

$$\mathbf{c}_o = \sum_{i=1}^{n} (\alpha_i \mathbf{c}_i) + \mathbf{c}_d (1 - \sum_{i=1}^{n} \alpha_i)$$

ILLINOIS

Multi-layer Blending: Weighted Average

$$\mathbf{c}_{\text{sum}} = \sum_{i=1}^{n} (\alpha_i \mathbf{c}_i), \quad \alpha_{\text{sum}} = \sum_{i=1}^{n} \alpha_i,$$

$$\mathbf{c}_{\text{wavg}} = \frac{\mathbf{c}_{\text{sum}}}{\alpha_{\text{sum}}}, \quad \alpha_{\text{avg}} = \frac{\alpha_{\text{sum}}}{n},$$

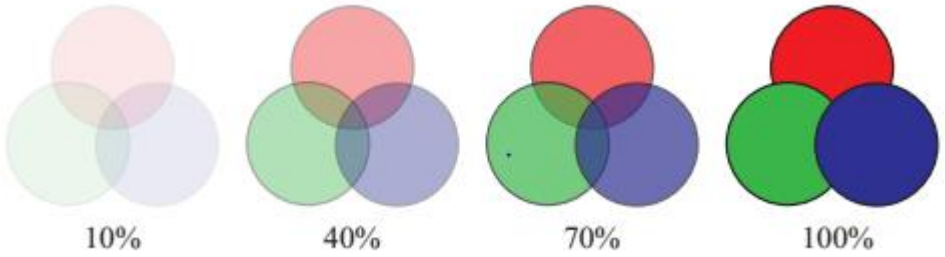
$$u = (1 - \alpha_{\text{avg}})^n,$$

$$\mathbf{c}_o = (1 - u)\mathbf{c}_{\text{wavg}} + u\mathbf{c}_d.$$

Avoids problems with weighted sum

- Saturated color values
- Alphas exceeding 1.0

Order is more important at higher opacity



ILLINOIS

Examples




In the upper left, traditional back-to-front alpha blending is performed, leading to rendering errors due to incorrect sort order. In the upper right, the A-buffer is used to give a perfect, non-interactive result. The lower left presents the rendering with multi-layer alpha blending. The lower right shows the differences between the A-buffer and multi-layer images.(Images courtesy of Marco Salvi and Karthik Vaidyanathan, Intel Corporation.)
Real-Time Rendering, Fourth Edition