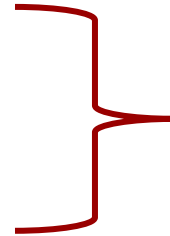Lecture 16

# Color and Textures

# Graphics Lectures

- ## Drawing Images
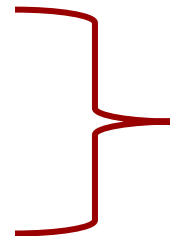  - ### SpriteBatch interface
  - ### Coordinates and Transforms

bare minimum
to draw graphics
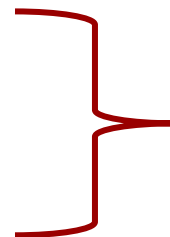
- ## Drawing Perspective
  - ### Camera
  - ### Projections

side-scroller vs.
top down

- ## **Drawing Primitives**
  - ### Color and Textures
  - ### Polygons

necessary for
lighting & shadows

the gamedesigninitiative
at cornell university

# Take Away For Today

- Image **color** and **composition**

  - What is the RGB model for images?

  - What does alpha represent?

  - How does alpha composition work?

- **Graphics primitives**

  - How do primitives differ from sprites?

  - How does LibGDX support primitives?

  - How do we combine sprites and primitives?

# Drawing Multiple Objects

- Objects are on a **stack**
  - Images are *layered*
  - Drawn in order given

- Uses **color composition**
  - Often just draws last image
  - What about **transparency**?

- We need to understand…
  - How color is represented
  - How colors combine

Color & Texture
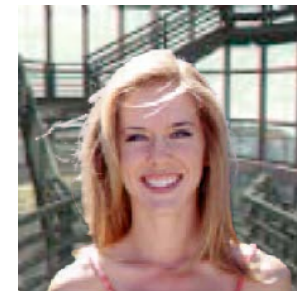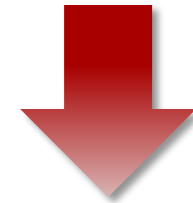
# Color Representation

- Humans are **Trichromatic**
  - Any color a blend of three
  - Images from only 3 colors

- Additive Color
  - Each color has an intensity
  - Blend by adding intensities

- Computer displays:
  - Light for each "channel"
  - Red, green and blue

- Aside: Subtractive Color
  - Learned in primary school
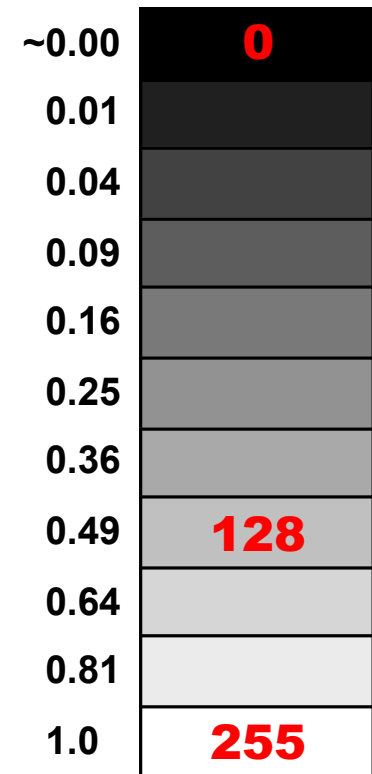  - For pigments, not light



[Cornell CS 465 Slides]

red          green          blue

Color & Texture

the **gamedesign**initiative
at cornell university

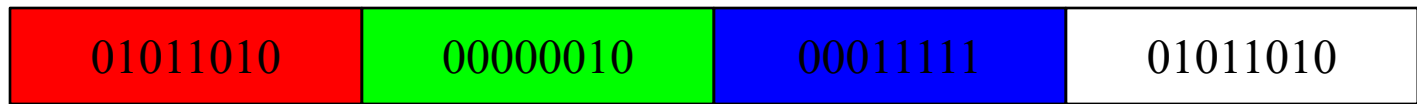# Color Blending Example

Color & Texture

# Color Representation

- Each color has an **intensity**
  - Measures amount of light of that color
  - 0 = absent, 1 = maximum intensity

- Real numbers take up a lot of space
  - **Compact representation**: one byte (0-255)
  - As good as human eye can distinguish

- But graphics algorithms require [0,1]
  - Use [0,255] for *storage only*
  - intensity = bits/255.0
  - bits = floor(intensity*255)

| Intensity | Byte |
|-----------|------|
| ~0.00 | 0 |
| 0.01 | |
| 0.04 | |
| 0.09 | |
| 0.16 | |
| 0.25 | |
| 0.36 | |
| 0.49 | 128 |
| 0.64 | |
| 0.81 | |
| 1.0 | 255 |

Color & Texture

# Color Representation

- Intensity for three colors: 3 bytes or 24 bits

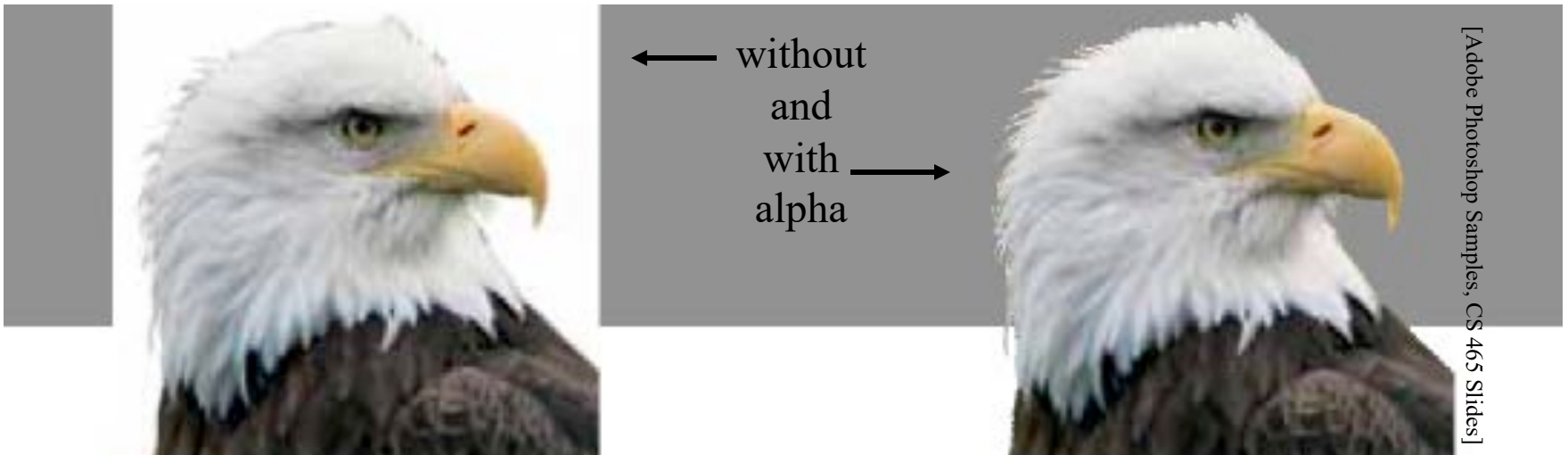| 01011010 | 00000010 | 00011111 | 01011010 |
|:---:|:---:|:---:|:---:|

**HTML Color**       #5A             02             1F          Not Supported

- Store as a 32 bit int; use bit ops to access
  - red:    `0x000000FF & integer`
  - green: `0x000000FF & (integer >> 8)`
  - blue:  `0x000000FF & (integer >> 16)`

- Most integers are actually 4 bytes; what to do?
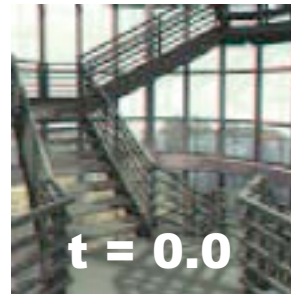
Color & Texture

# The Alpha Channel

- Only used in **color composition**

- Does *not* correspond to a physical light source
  - Allows for transparency of overlapping objects
  - Without it the colors are written atop another



without
and
with
alpha

[Adobe Photoshop Samples, CS 465 Slides]

Color & Texture

the
gamedesigninitiative
at cornell university

# Color Composition

- Trivial example: Video crossfade
  - Smooth transition from one scene to another.


[Chuang et al/ Corel]

A    B    t = 0.0

$$r_C = tr_A + (1-t)r_B$$
$$g_C = tg_A + (1-t)g_B$$
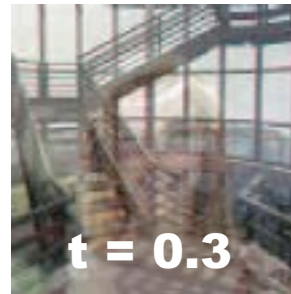$$b_C = tb_A + (1-t)b_B$$

per pixel calculation

- Note sums weight to 1.0
  - No unexpected brightening or darkening
  - No out-of-range results
- This is an example of **linear interpolation**

Color & Texture

the **gamedesigninitiative**
at cornell university

# Color Composition

- Trivial example: Video crossfade
  - Smooth transition from one scene to another.



[Chuang et al/ Corel]

A  B  t = 0.3

$$r_C = tr_A + (1 - t)r_B$$
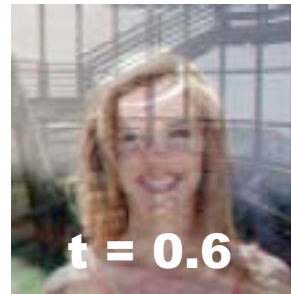$$g_C = tg_A + (1 - t)g_B$$
$$b_C = tb_A + (1 - t)b_B$$

per pixel calculation

- Note sums weight to 1.0
  - No unexpected brightening or darkening
  - No out-of-range results
- This is an example of **linear interpolation**

Color & Texture

the gamedesigninitiative
at cornell university

# Color Composition

- Trivial example: Video crossfade
  - Smooth transition from one scene to another.


[Chuang et al/ Corel]

A    B    t = 0.6

$$r_C = tr_A + (1 - t)r_B$$
$$g_C = tg_A + (1 - t)g_B$$
$$b_C = tb_A + (1 - t)b_B$$

per pixel calculation

- Note sums weight to 1.0
  - No unexpected brightening or darkening
  - No out-of-range results
- This is an example of **linear interpolation**

Color & Texture

the game**design**initiative
at cornell university

# Color Composition

- Trivial example: Video crossfade
  - Smooth transition from one scene to another.



[Chuang et al/ Corel]

A      B      t = 0.8

$$r_C = tr_A + (1 - t)r_B$$
$$g_C = tg_A + (1 - t)g_B$$
$$b_C = tb_A + (1 - t)b_B$$

per pixel calculation

- Note sums weight to 1.0
  - No unexpected brightening or darkening
  - No out-of-range results
- This is an example of **linear interpolation**

the gamedesigninitiative
at cornell university

# Color Composition

- Trivial example: Video crossfade
  - Smooth transition from one scene to another.



[Chuang et al/ Corel]

A     B     t = 1.0

$$r_C = tr_A + (1-t)r_B$$
$$g_C = tg_A + (1-t)g_B$$
$$b_C = tb_A + (1-t)b_B$$

per pixel calculation

- Note sums weight to 1.0
  - No unexpected brightening or darkening
  - No out-of-range results
- This is an example of **linear interpolation**

the gamedesigninitiative
at cornell university

# Foreground and Background

- In many cases, just adding is not enough
  - Want some elements in composite, not others
  - Do not want transparency of crossfade

- How we compute new image varies with position.



Use background

Use foreground
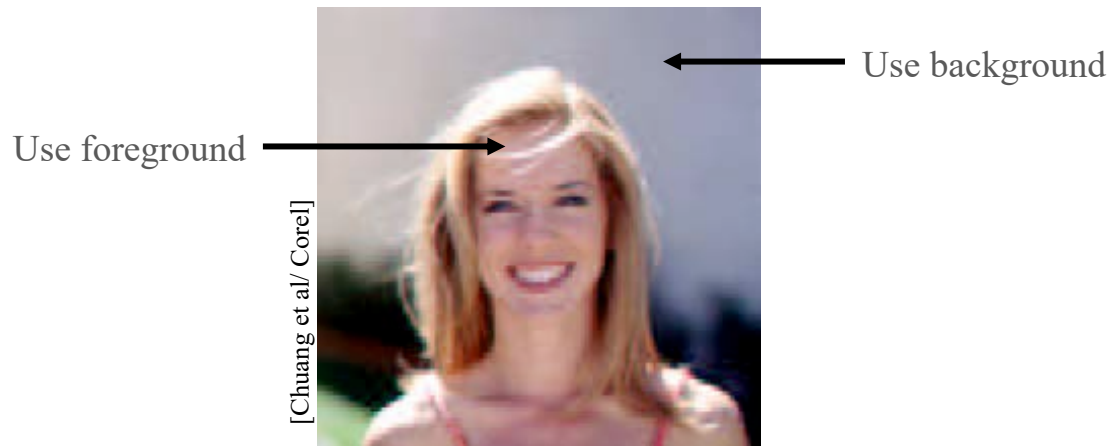
[Chuang et al/ Corel]

- Need to store a tag indicating parts of interest

the gamedesigninitiative
at cornell university

# Binary Image Mask

- First idea: Store one bit per pixel

  - Answers question "Is this pixel in foreground?"



[Chuang et al/ Corel]  [Cornell PCG]

  - Does not work well near the edges

the gamedesigninitiative
at cornell university

# Binary Image Mask

- First idea: Store one bit per pixel
  - Answers question "Is this pixel in foreground?"



[Chuang et al/ Corel]    [Cornell PCG]

  - Does not work well near the edges

Color & Texture

the gamedesigninitiative
at cornell university

# Binary Image Mask

- First idea: Store one bit per pixel
  - Answers question "Is this pixel in foreground?"



[Chuang et al/ Corel]    [Cornell PCG]

  - Does not work well near the edges
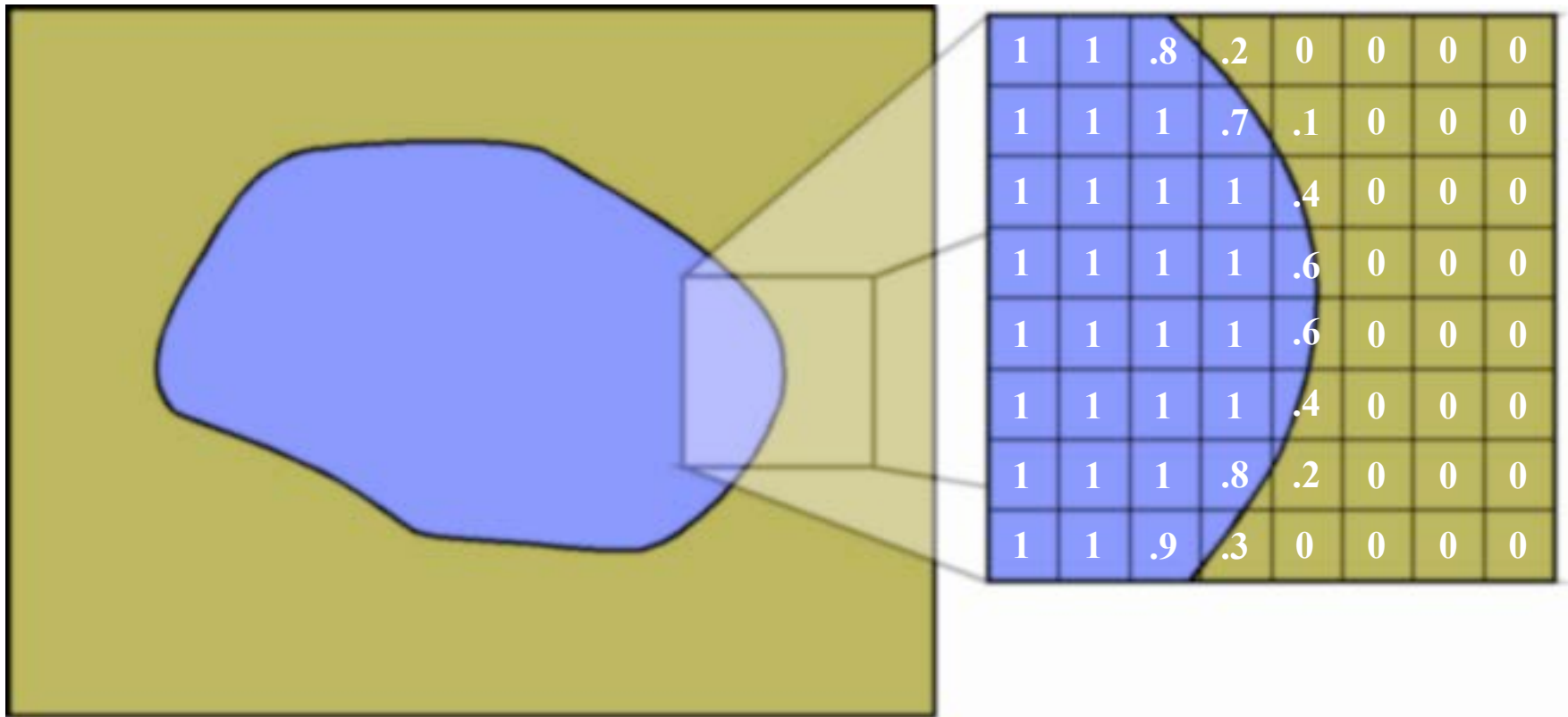
Color & Texture
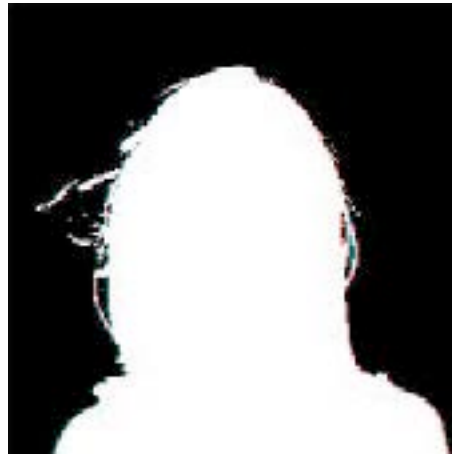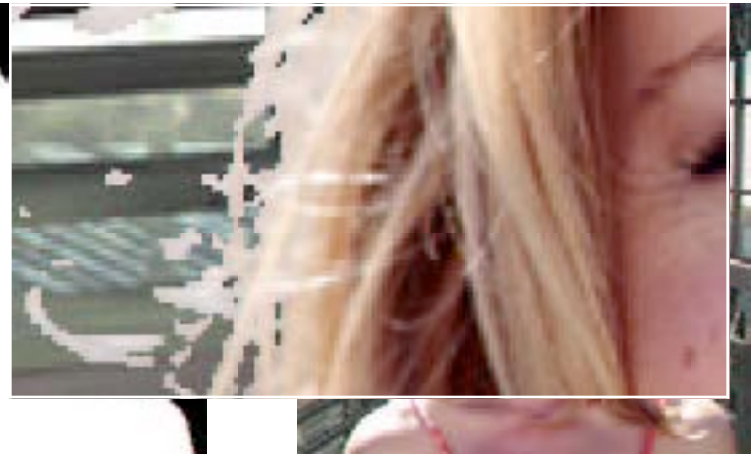
# Partial Pixel Coverage

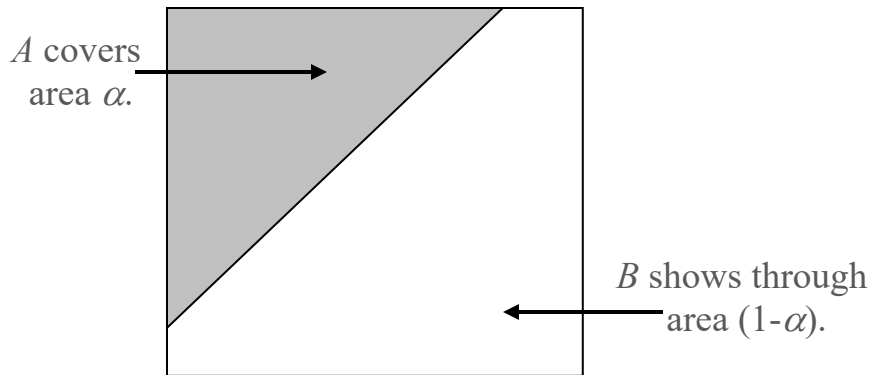**Problem**: Boundary neither foreground nor background
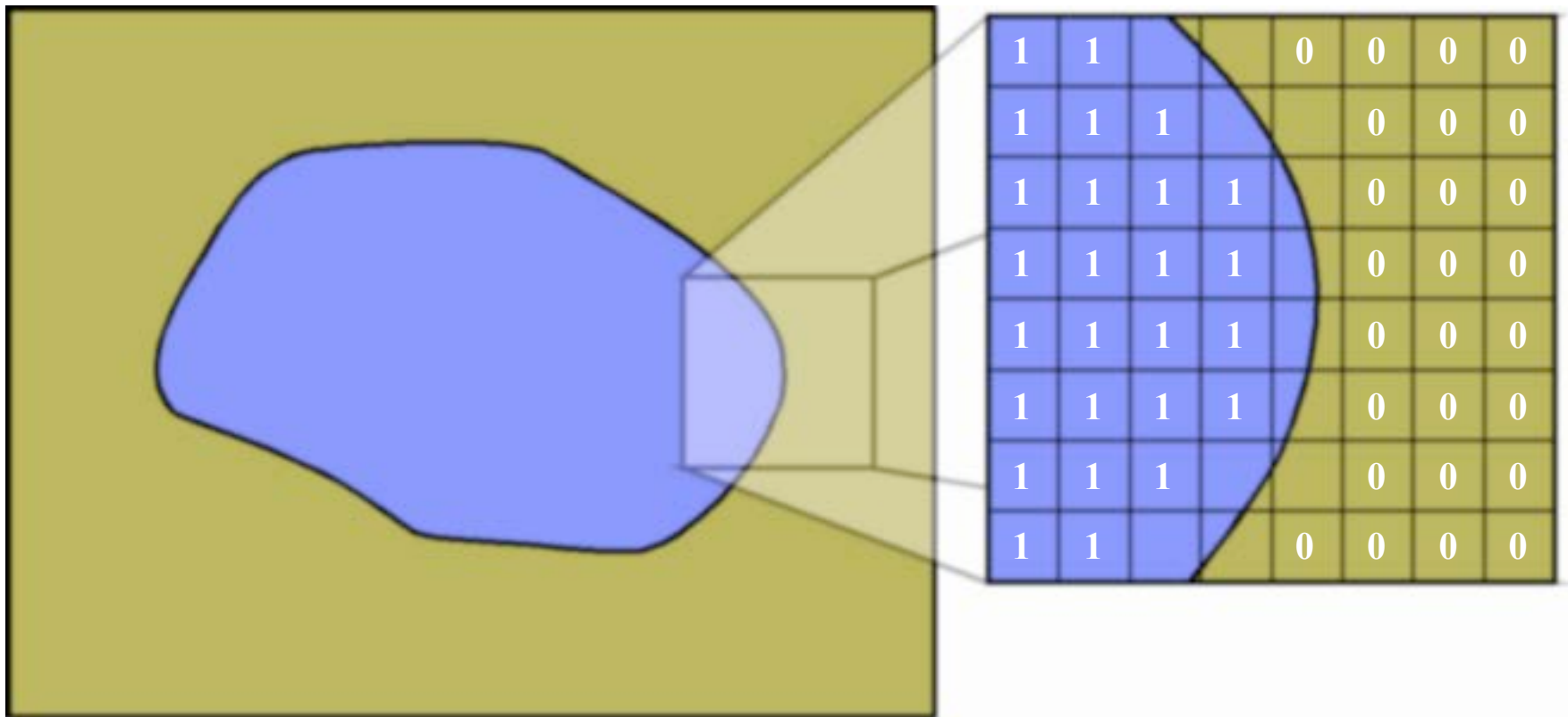
# Partial Pixel Coverage

**Solution**: Interpolate on the border (Not exact, but *fast*)

Color & Texture

# Alpha Compositing

- Formalized in 1984 by Porter & Duff

- **Store fraction of pixel covered**; call it $\alpha$

*A* covers area $\alpha$.

*B* shows through area $(1-\alpha)$.

$$C = A \textbf{ over } B$$

$$r_C = \alpha_A r_A + (1 - \alpha_A) r_B$$

$$g_C = \alpha_A g_A + (1 - \alpha_A) g_B$$

$$b_C = \alpha_A b_A + (1 - \alpha_A) b_B$$

- Clean implementation; 8 more bits makes 32
  - 2 multiplies + 1 add for compositing

# Alpha Compositing Example

- Repeat previous with grey scale mask
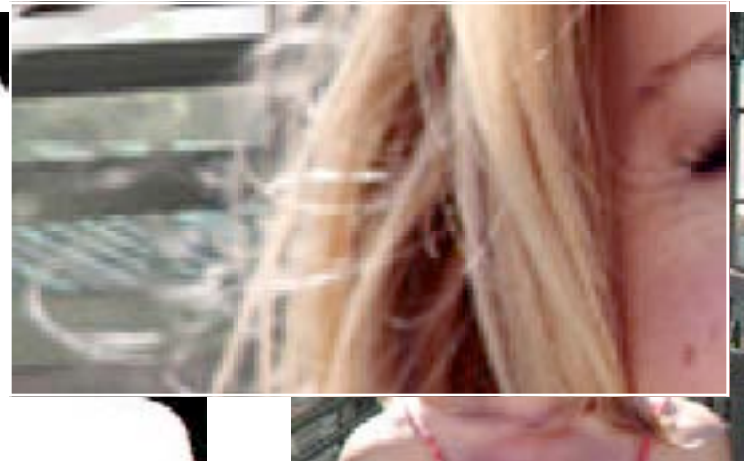  - Edges are much better now



[Chuang et al/ Corel]    [Cornell PCG]

Color & Texture

# Alpha Compositing Example

- Repeat previous with grey scale mask
  - Edges are much better now



[Chuang et al/ Corel]    [Cornell PCG]

the game**design**initiative
at cornell university
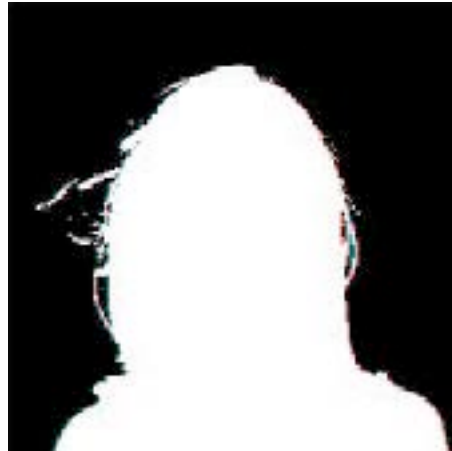
# Compositing in LibGDX

- spriteBatch.setBlendFunction(src, dst);

  $$\underbrace{\qquad\qquad}_{\text{OpenGL Constants}}$$

  - **General Formula**: $c_C = (\text{src})c_A + (\text{dst})c_B$

- **Alpha Blending**
  - src = GL20.GL_SRC_ALPHA $\qquad\qquad (a_A)$
  - dst = GL20.GL_ONE_MINUS_SRC_ALPHA $\quad (1\text{-}a_A)$

- Colors may be **premultiplied**: $c' = ca$
  - src = GL20.GL_ONE
  - dst = GL20.GL_ONE_MINUS_SRC_ALPHA

Color & Texture

# Compositing in LibGDX

- spriteBatch.setBlendFunction(<span style="color:blue">src</span>, <span style="color:red">dst</span>);

  OpenGL Constants

  - **General Formula**: $c_C = (\text{src})c_A + (\text{dst})c_B$

- **Additive Blending** (not premultiplied)
  - <span style="color:blue">src</span> = GL20.GL_SRC_ALPHA
  - <span style="color:red">dst</span> = GL20.GL_ONE

- **Opaque** (no blending at all)
  - <span style="color:blue">src</span> = GL20.GL_ONE
  - <span style="color:red">dst</span> = GL20.GL_ZERO

Color & Texture
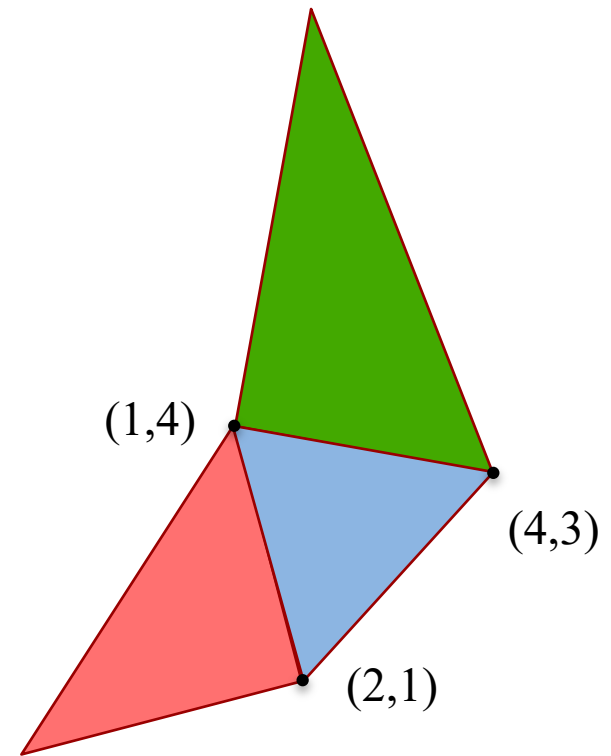
# The Problem with Sprites

- ## Sprites drawn by artist
  - Distort with transforms
  - Major changes require new art from artist
  - Inefficient collaboration

- ## Sprite-free graphics?
  - Simple geometries
  - Particle effects
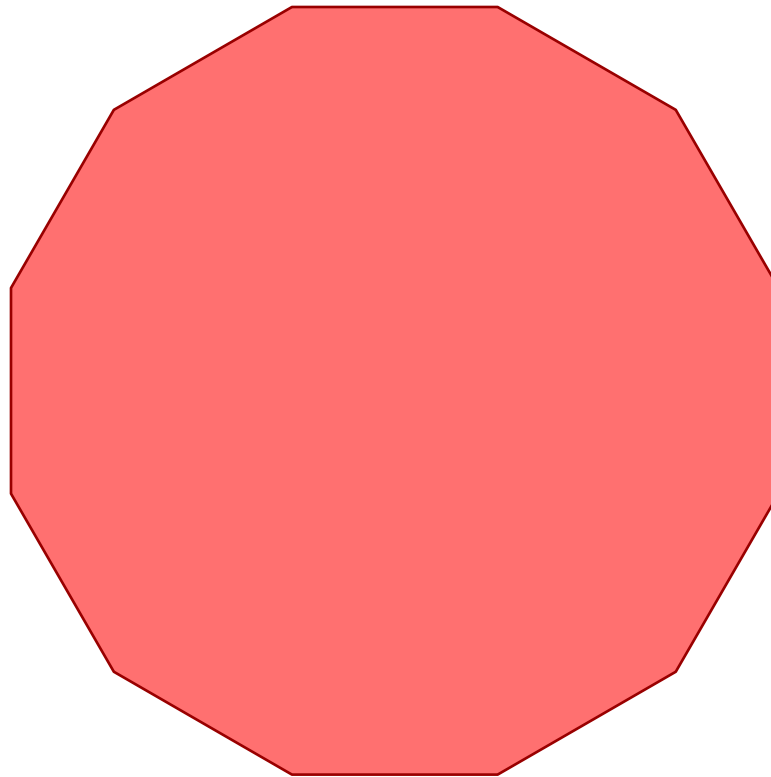  - Dynamic shapes
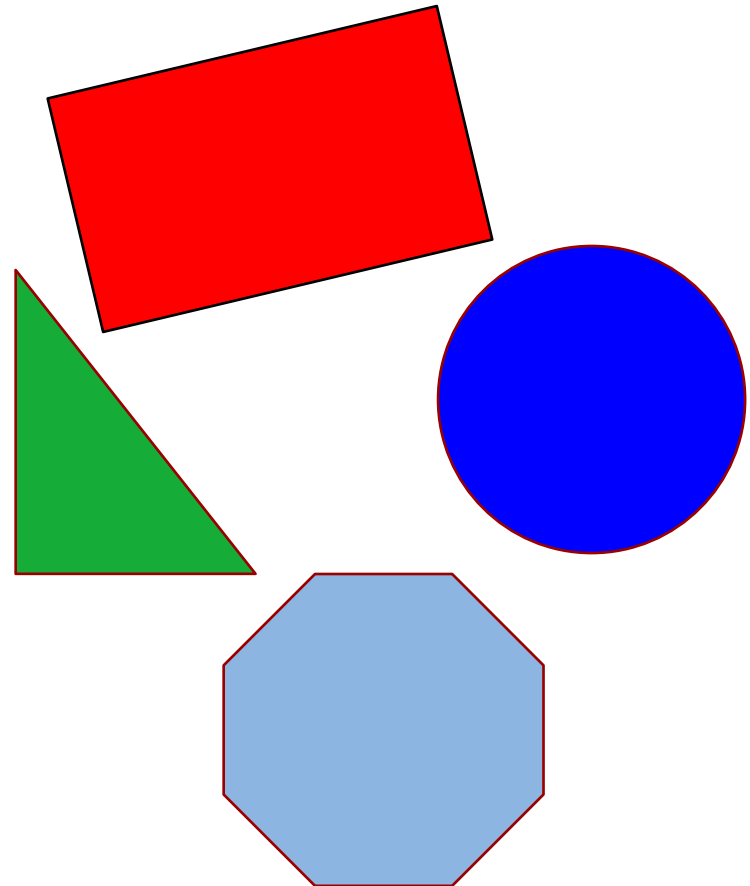
# Triangles in Computer Graphics

- Everything made of **triangles**
  - Mathematically "nice"
  - Hardware support (GPUs)

- Specify with **three vertices**
  - Coordinates of corners

- Composite for complex shapes
  - Array of vertex objects
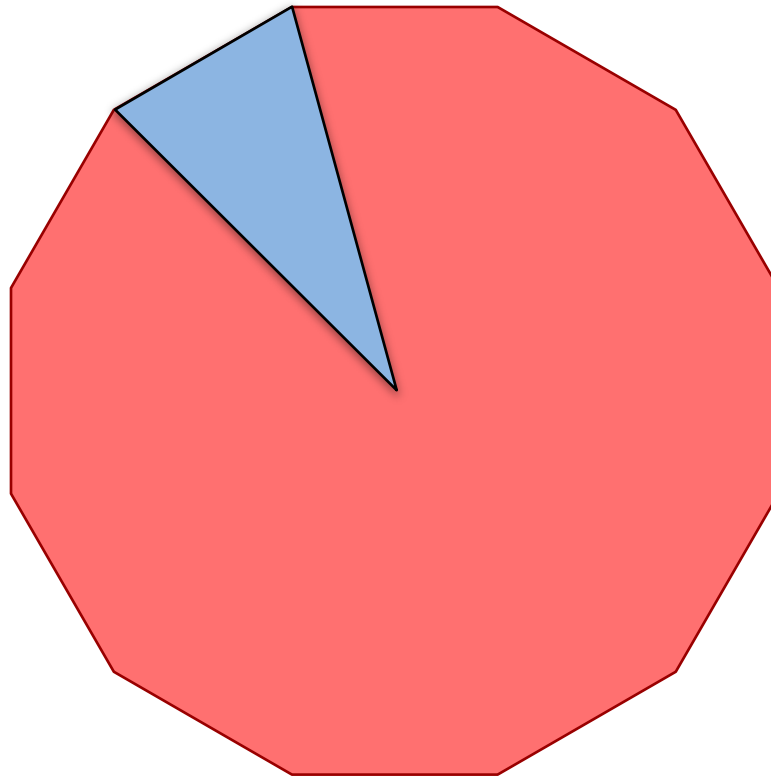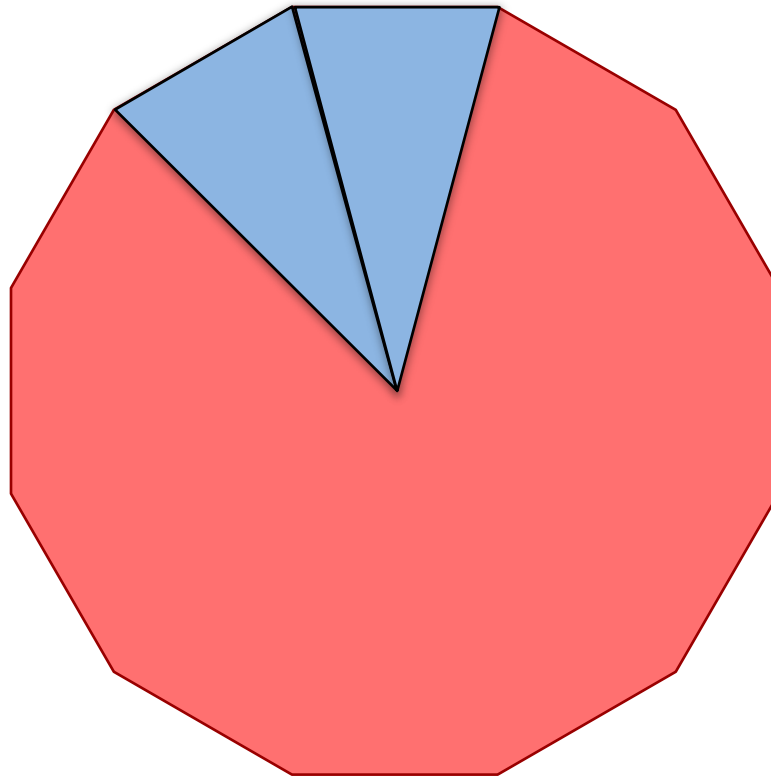  - Each 3 vertices = triangle

(1,4)

(4,3)

(2,1)

Color & Texture

# Triangulation of Polygons

Color & Texture

# Triangulation of Polygons

Color & Texture

# Triangulation of Polygons

Color & Texture

# Triangulation of Polygons

Color & Texture

# Round Shapes?

Color & Texture

# Round Shapes?

Color & Texture

# ShapeRenderer in LibGDX

- Tool to draw triangles
  - Specify a general shape
  - Makes the triangles for you

- Works like a SpriteBatch
  - Has a begin/end
  - Can set default color
  - Several draw commands

- Can mix with SpriteBatch
  - But not at the same time!
  - End one before begin other

```
render.circle(200, 200, 100, 5);
```

Color & Texture

the gamedesigninitiative
at cornell university
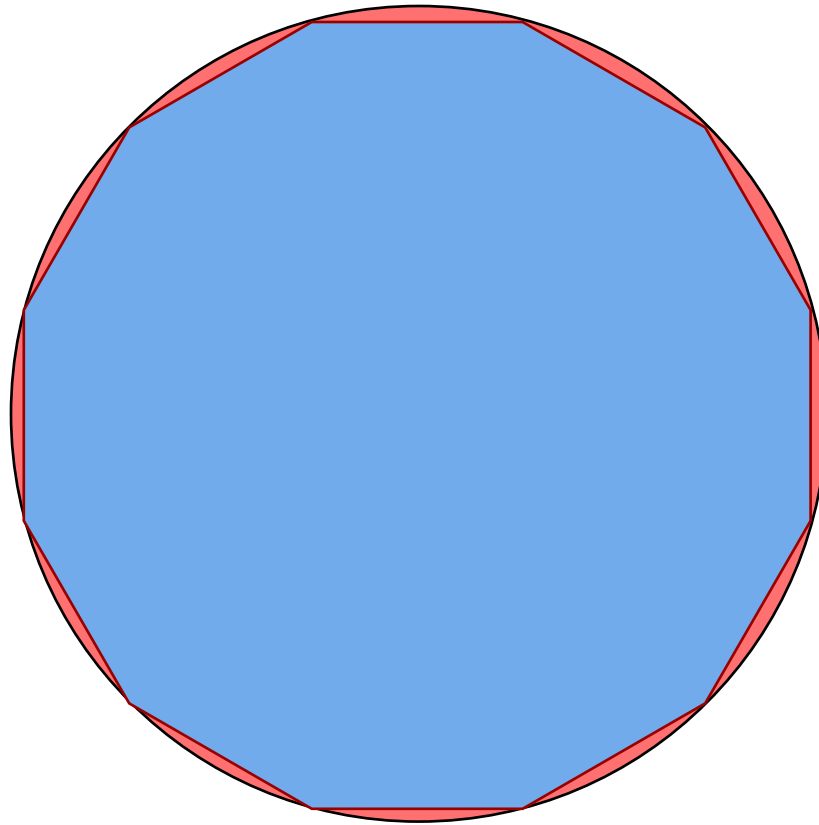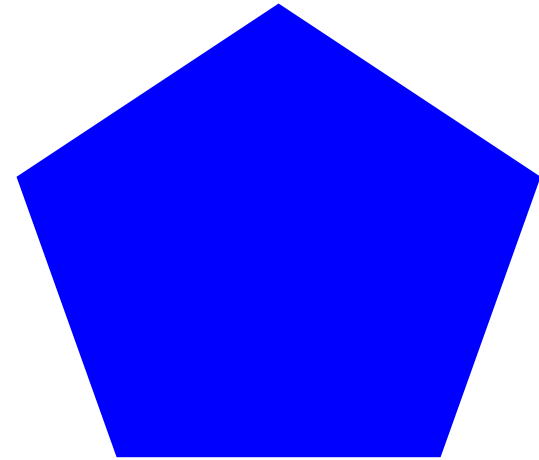
# ShapeRenderer in LibGDX

- Tool to draw triangles
  - Specify a general shape
  - Makes the triangles for you

- Works like a SpriteBatch
  - Has a begin/end
  - Can set default color
  - Several draw commands

- Can mix with SpriteBatch
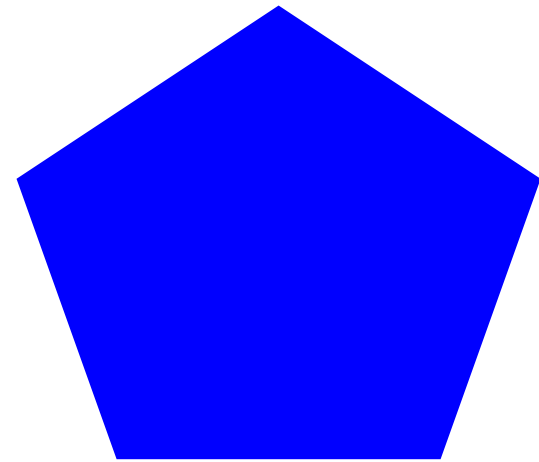  - But not at the same time!
  - End one before begin other

`render.circle(200, 200, 100, 5)`

Number of triangles

Color & Texture

# ShapeRenderer Example

```
render.begin(ShapeRenderer.ShapeType.Filled);
render.setColor(Color.BLUE);
render.circle(200, 200, 100, 8);
render.end();


render.begin(ShapeRenderer.ShapeType.Line);
render.setColor(Color.RED);
render.circle(200, 200, 100, 8);
render.end();
```

Color & Texture

the gamedesigninitiative
at cornell university

# ShapeRenderer Example

```
render.begin(ShapeRenderer.ShapeType.Filled);
render.setColor(Color.BLUE);
render.circle(200, 200, 100, 8);
render.end();


render.begin(ShapeRenderer.ShapeType.Line);
render.setColor(Color.RED);
render.circle(200, 200, 100, 8);
render.end();
```

Color & Texture

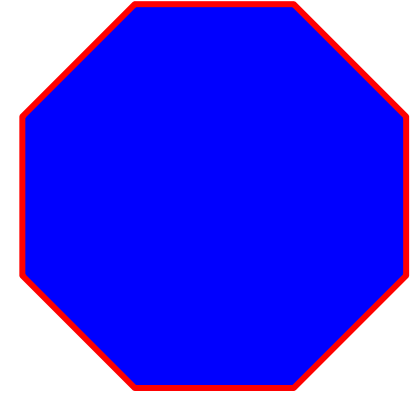the gamedesigninitiative
at cornell university

# ShapeRenderer Example

```
render.begin(ShapeRenderer.ShapeType.Filled);
render.setColor(Color.BLUE);
render.circle(200, 200, 100, 8);
render.end();


render.begin(ShapeRenderer.ShapeType.Line);
render.setColor(Color.RED);
render.circle(200, 200, 100, 8);
render.end();
```
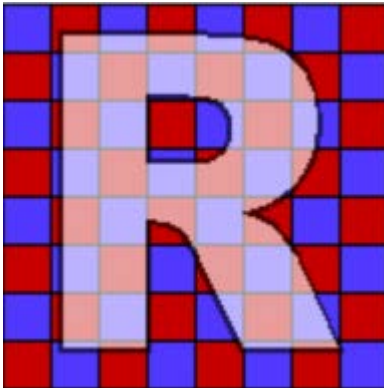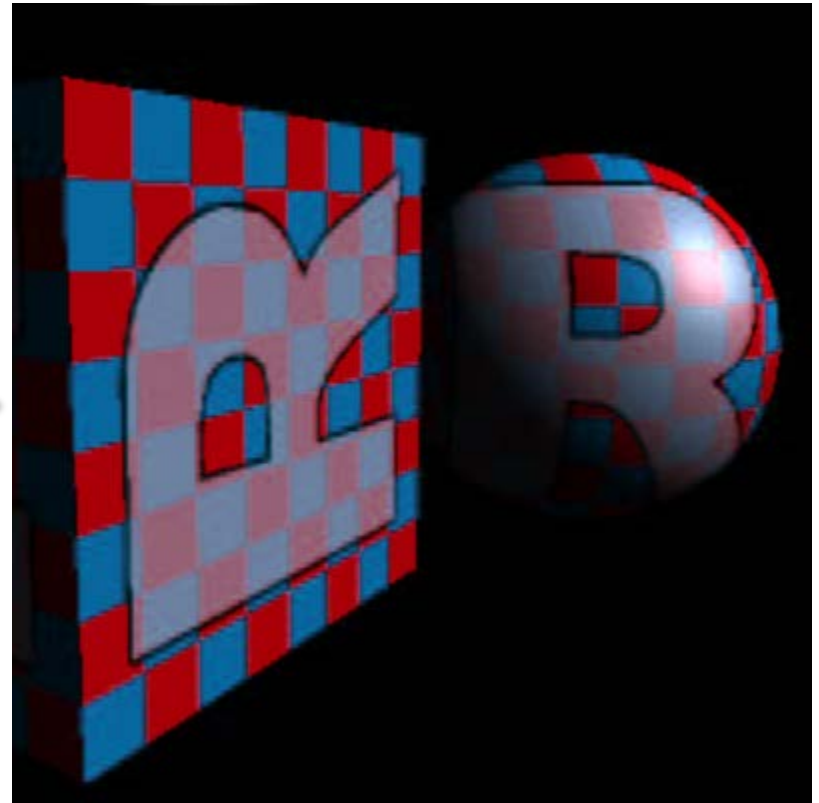
Note separate pass for filled, outline

the
gamedesigninitiative
at cornell university

# Textures

2D Image File



Mapped On To
Polygonal Shape

# Simple Texturing in LibGDX

- **PolygonSpriteBatch** handles 90% of all cases
  - Works like a normal `SpriteBatch`
  - But now specify image and polygon
  - Entirely replaced SpriteBatch in **Lab 4**

- Uses the **PolygonRegion** class
  - Way to specify what part of image to use
  - Specify as a collection of vertices
  - Specify using **pixel positions**, not **texture coords**
  - See `PolygonObstacle` in Lab 4

Color & Texture

# PolygonRegion Example



(192,128)

(0,64)

(0,0)    (128,0)
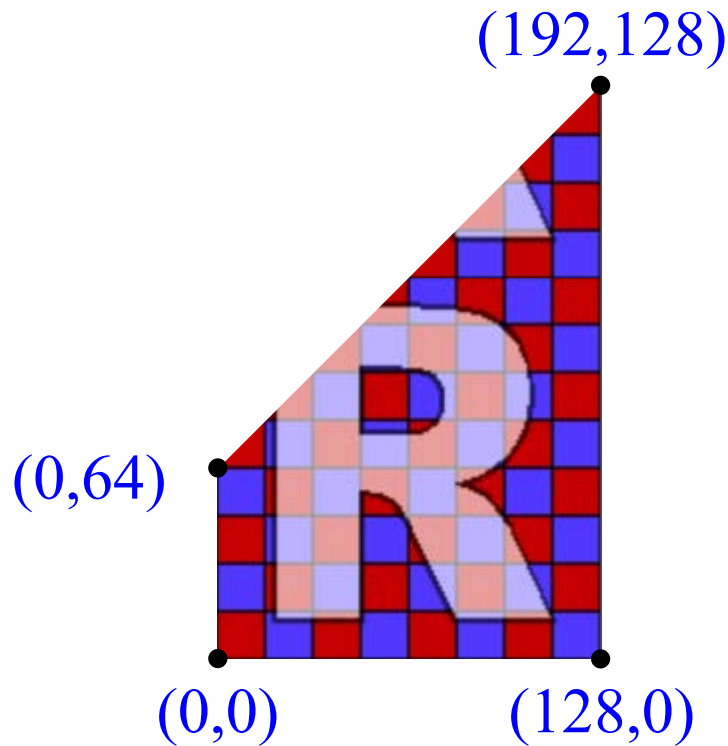
verts = {0,0,0,64,192,128,128,0}

- Create vertices by **pixel pos**
  - Example texture is 124x124
  - Preferences set to wrap
  - Store as an array of floats

- Must convert into triangles
  - Each vertex has an index
  - Given by position in array
  - Create array of indices

- Construct `PolygonRegion`
  - Specify texture
  - Specify vertices+triangles

Color & Texture

the gamedesigninitiative
at cornell university

# PolygonRegion Example

Index 2
(192,128)

(0,64)
Index 1

(0,0)
Index 0

(128,0)
Index 3

```
verts = {0,0,0,64,192,128,128,0}
  tris = {0,1,3,3,1,2}
```

- Create vertices by pixel pos
  - Example texture is 124x124
  - Preferences set to wrap
  - Store as an array of floats

- Must **convert into triangles**
  - Each vertex has an index
  - Given by position in array
  - Create array of indices

- Construct PolygonRegion
  - Specify texture
  - Specify vertices+triangles

42

Color & Texture
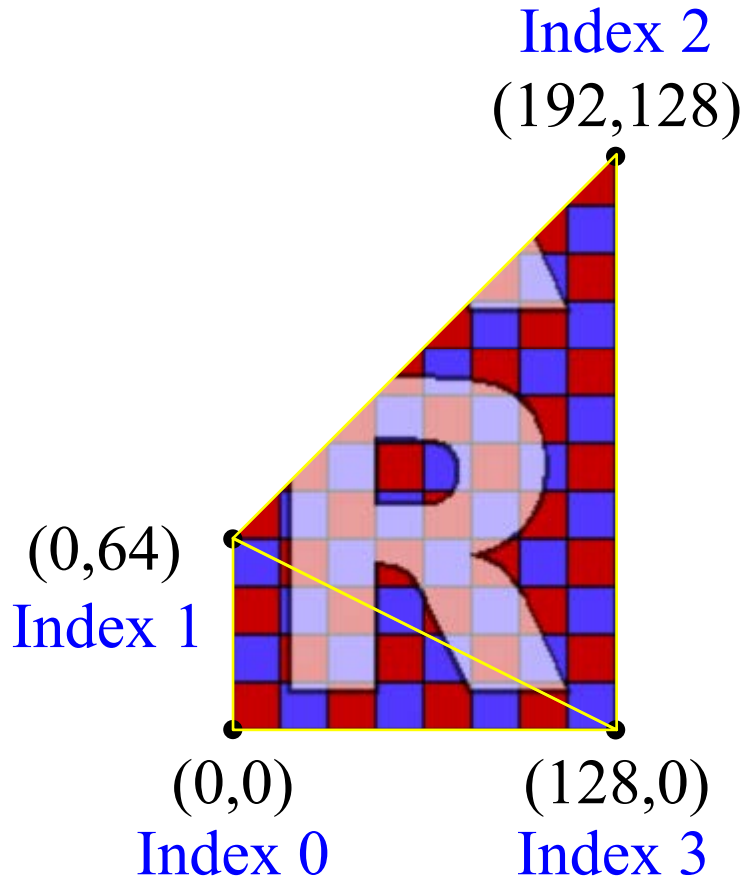
# PolygonRegion Example

new PolygonRegion(img,verts,tris)
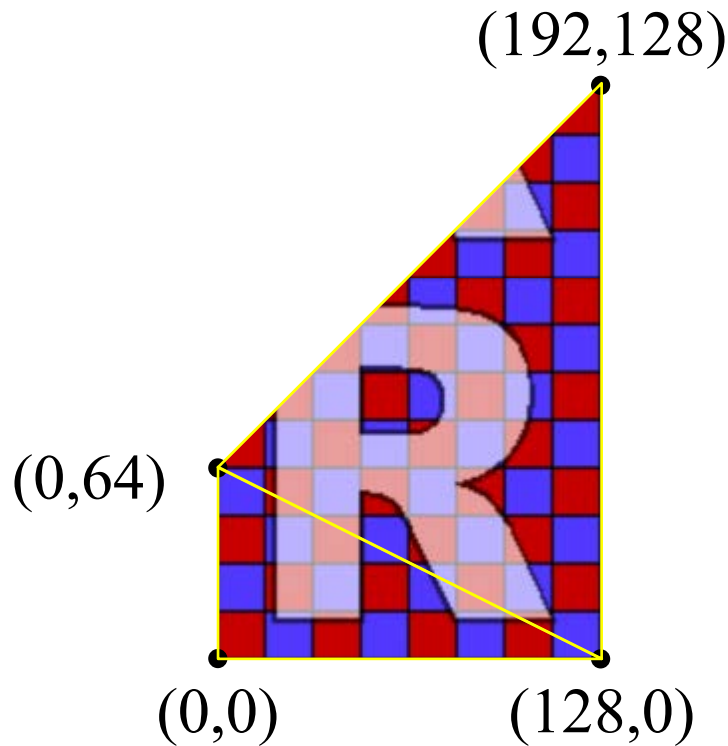


(192,128)

(0,64)

(0,0)  (128,0)

verts = {0,0,0,64,192,128,128,0}
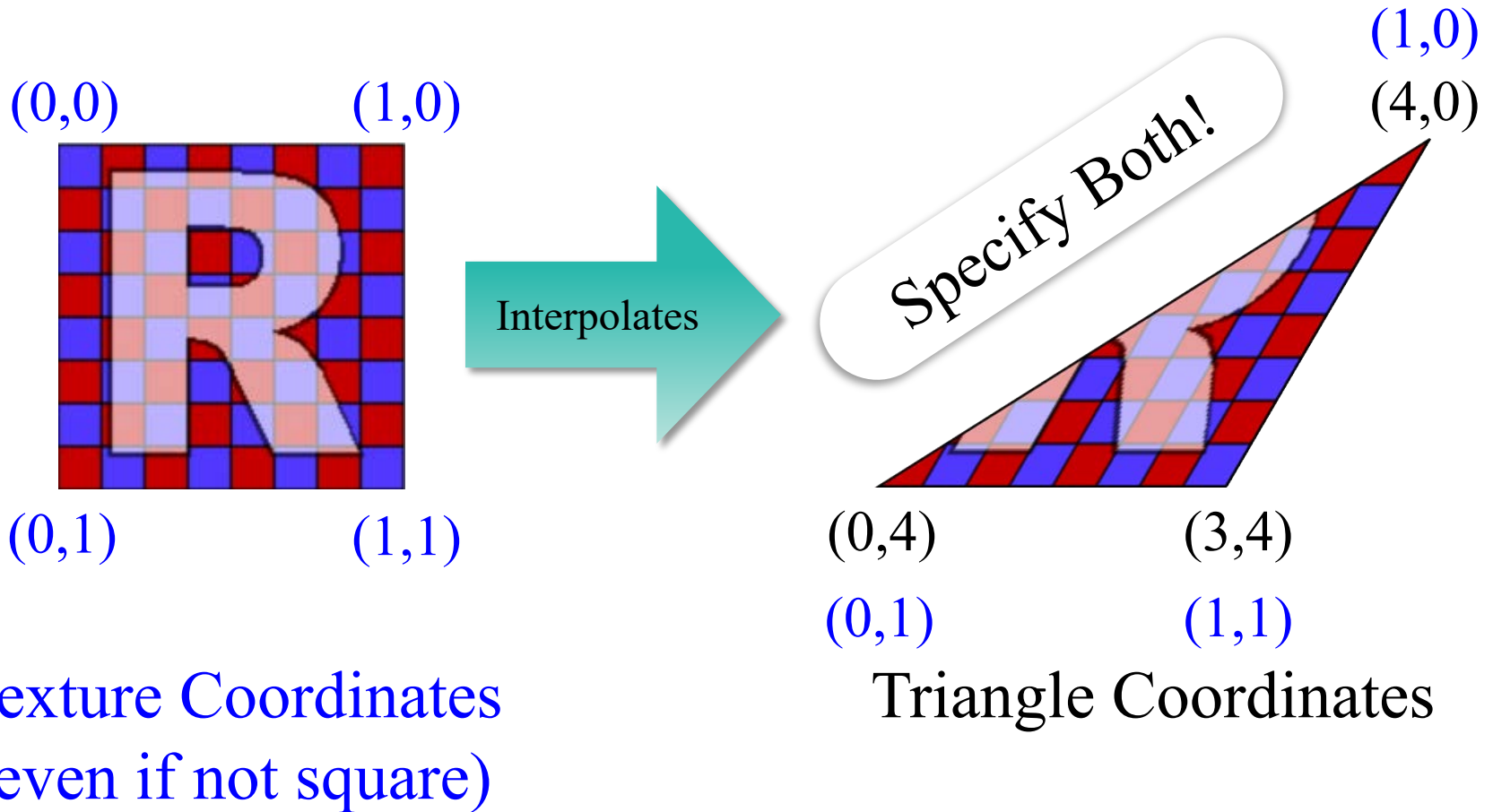  tris = {0,1,3,3,1,2}

- Create vertices by pixel pos
  - Example texture is 124x124
  - Preferences set to wrap
  - Store as an array of floats

- Must convert into triangles
  - Each vertex has an index
  - Given by position in array
  - Create array of indices

- **Construct PolygonRegion**
  - Specify texture
  - Specify vertices+triangles

Color & Texture

# What If I Know OpenGL?

- Use the **GL20** (OpenGLES 2.0) object
  - Standard OpenGL functions are its methods
  - Standard OpenGL values are its constants

- There is a **GL30** (OpenGLES 3.0), but
  - It is not the default OpenGL in LibGDX
  - Requires special `DesktopLauncher` settings

- See **Programming Lab 2** for examples
  - Uses a custom OpenGL shader
  - Also advanced LibGDX classes like Mesh

Color & Texture

the game**design**initiative
at cornell university

# OpenGL Texturing

(0,0)                    (1,0)

**R**

(0,1)                    (1,1)

Interpolates

Specify Both!

(1,0)
(4,0)

(0,4)                    (3,4)

(0,1)                    (1,1)

**Texture Coordinates**
**(even if not square)**

**Triangle Coordinates**

# OpenGL Texturing



(0,0)        (1,0)

(0,1)        (1,1)

Interpolates

(0,0)        (1,0)

(2,0)        (4,0)

(0,4)        (3,4)

(0,1)        (1,1)
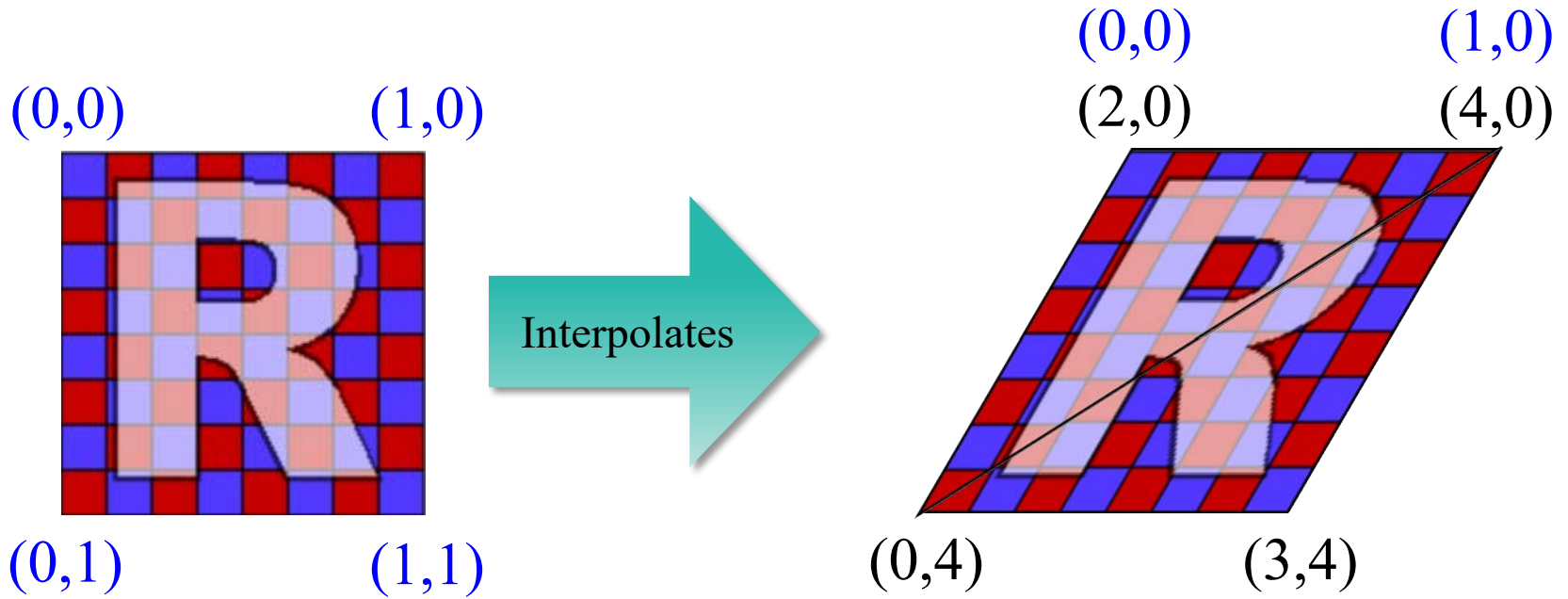
**Texture Coordinates
(even if not square)**

**Triangle Coordinates
(more than one triangle)**

Color & Texture

the
game**design**initiative
at cornell university

# Summary

- Computer images defined by **color channels**
  - Three visible channels: red, green, blue

- Sprites combined via **compositing**
  - Alpha = percentage color in foreground

- Can use **triangles** instead of sprites
  - Complex shapes defined by arrays of triangles

- **Textures** generalize the notion of color
  - 2D image that is used to "color" triangle
  - Need triangle coordinates **and** texture coordinates