Lecture 17

Color and Textures
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?
  - What graphics primitives do XNA support?
  - 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 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

---

Color & Texture
Color Blending Example

Color Blending Example

blue

red

green
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)
Color Representation

- Intensity for three colors: 3 bytes or 24 bits

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

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

HTML Color

| #5A | 02 | 1F | Not Supported |
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

[Adobe Photoshop Samples, CS 465 Slides]
Color Composition

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

  ![A](img1.png) ![B](img2.png) ![t = 0.0](img3.png)

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

  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 Composition

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

  ![Images of A and B with t = 0.3]
  \[
  \begin{align*}
  r_C &= tr_A + (1 - t)r_B \\
  g_C &= tg_A + (1 - t)g_B \\
  b_C &= tb_A + (1 - t)b_B
  \end{align*}
  \]

  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 Composition**

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

- Note sums weight to 1.0
  - No unexpected brightening or darkening
  - No out-of-range results

- This is an example of **linear interpolation**

\[
\begin{align*}
    r_C &= tr_A + (1 - t)r_B \\
    g_C &= tg_A + (1 - t)g_B \\
    b_C &= tb_A + (1 - t)b_B
\end{align*}
\]

per pixel calculation

[Chuang et al/ Corel]
Trivial example: Video crossfade
- Smooth transition from one scene to another.

- Note sums weight to 1.0
  - No unexpected brightening or darkening
  - No out-of-range results

- This is an example of **linear interpolation**

\[
\begin{align*}
  r_C &= tr_A + (1 - t)r_B \\
  g_C &= tg_A + (1 - t)g_B \\
  b_C &= tb_A + (1 - t)b_B
\end{align*}
\]
Color Composition

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

  ![Image of A, B, and C with t = 0.0, 0.3, 0.6, 0.8]

  \[
  \begin{align*}
  r_C &= tr_A + (1 - t)r_B \\
  g_C &= tg_A + (1 - t)g_B \\
  b_C &= tb_A + (1 - t)b_B
  \end{align*}
  \]

  per pixel calculation

  • Note sums weight to 1.0
  • No unexpected brightening or darkening
  • No out-of-range results

  • This is an example of \textbf{linear interpolation}
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.

- Need to store a tag indicating parts of interest
Binary Image Mask

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

- Does not work well near the edges
Binary Image Mask

- First idea: Store one bit per pixel
  - Answers question “Is this pixel in foreground?”
  - Does not work well near the edges
Binary Image Mask

- First idea: Store one bit per pixel
  - Answers question “Is this pixel in foreground?”
  - Does not work well near the edges
Partial Pixel Coverage

Problem: Boundary neither foreground nor background

![Diagram showing partial pixel coverage problem]

Color & Texture
Partial Pixel Coverage

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

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.8</td>
<td>.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>.7</td>
<td>.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.8</td>
<td>.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>.9</td>
<td>.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Color & Texture
Alpha Compositing

- Formalized in 1984 by Porter & Duff
- **Store fraction of pixel covered**; call it $\alpha$

\[
C = A \text{ over } B
\]

\[
\begin{align*}
  r_C &= \alpha_A r_A + (1 - \alpha_A) r_B \\
  g_C &= \alpha_A g_A + (1 - \alpha_A) g_B \\
  b_C &= \alpha_A b_A + (1 - \alpha_A) b_B
\end{align*}
\]

- 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
Alpha Compositing Example

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

- `batch.Begin(spriteSort, blendState, ...)`

- **SpriteSortMode**: Draws the objects in order
  - **BackToFront**: The standard order
  - **FrontToBack**: Reverse; bottom over top

- **BlendState**: How to combine colors
  - **Opaque**: Colors drawn atop one another
  - **Additive**: The colors are added together
  - **AlphaBlend**: Composite; top over bottom
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)  (2,1)  (4,3)
Triangulation of Polygons
Round Shapes?
Round Shapes?
Triangles/Polygons in XNA

- **GraphicsDevice** class
  - DrawIndexedPrimitives method
  - Feed an array of vertices

- **VertexPositionColor** class
  - Vertex coords + Color value
  - Determines color at corner
  - Colors do not match?

- See Board.cs in Lab 2
  - But that has other details…
  - Demo for today’s lecture
Textures

2D Image File

Mapped On To Polygonal Shape
Textures and Triangles

Texture Coordinates
(even if not square)

Triangle Coordinates

Specify Both!

Interpolates
Textures and Triangles

Texture Coordinates
(even if not square)

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

- Contains vertex **position** information
  - Just like `VertexPositionColor` class
  - Also works with `DrawIndexedPrimitives`

- Also contains **texture coordinates**
  - Location in active texture to use

- But does not contain **texture**!
  - Cannot have different textures at corners
  - To enforce this, texture goes elsewhere
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