CS130 : Computer Graphics

Tamar Shinar
Computer Science & Engineering
UC Riverside
Raster Devices and Images
Raster Devices
Raster Display

Hearn, Baker, Carithers
Transmissive vs. Emissive Display

[H&B, Fig. 2.16]
Raster Display

red, green, blue subpixels
What is an image?

Continuous image

\[ I : \mathbb{R} \rightarrow V \]
\[ R \subset \mathbb{R}^2 \]
\[ V = \mathbb{R}^+ \quad \text{(grayscale)} \]
\[ V = (\mathbb{R}^+)^3 \quad \text{(color)} \]
A **raster image** is a 2D array storing pixel values at each pixel.
What is an image?

**Raster image**

\[ I : \mathbb{R} \rightarrow V \]

\[ R \subset \mathbb{Z}^2 \]

\[ V = \mathbb{R}^+ \quad \text{(grayscale)} \]

\[ V = (\mathbb{R}^+)^3 \quad \text{(color)} \]

Each pixel value represents the **average color** of the image over that pixel’s area.

\[ [-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5] \]

\[ n_x = \text{number of columns} \]

\[ n_y = \text{number of rows} \]
What is an image?

**Raster image**

$I : R \to V$

$R \subset \mathbb{Z}^2$

$V = [0, 1]$  \hspace{0.5cm} \text{(grayscale)}

$V = [0, 1]^3$  \hspace{0.5cm} \text{(color)}

Each pixel value represents the **average color** of the image over that pixel’s area.

$$[-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$$

$n_x = \text{number of columns}$

$n_y = \text{number of rows}$
Color Representation

additive RGB

subtractive CMYK
Color Representation

sRGB color triangle

comparison of color gamuts

[wikipedia]
Bit depth - defined by device standards

<table>
<thead>
<tr>
<th>Bit-Depth</th>
<th>Number of Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 (monochrome)</td>
</tr>
<tr>
<td>2</td>
<td>4 (CGA)</td>
</tr>
<tr>
<td>4</td>
<td>16 (EGA)</td>
</tr>
<tr>
<td>8</td>
<td>256 (VGA)</td>
</tr>
<tr>
<td>16</td>
<td>65,536 (High Color, XGA)</td>
</tr>
<tr>
<td>24</td>
<td>16,777,216 (True Color, SVGA)</td>
</tr>
<tr>
<td>32</td>
<td>16,777,216 (True Color + Alpha Channel)</td>
</tr>
</tbody>
</table>

(Note alpha)

(Humans can perceive ~10,000,000 colors)
Alpha Channel

\[ \mathbf{c} = \alpha \mathbf{c}_f + (1 - \alpha) \mathbf{c}_b \]
Graphics Pipeline
Modern graphics system

[Angel and Shreiner]
Z-buffer Rendering

- Z-buffering is a very common approach, also often accelerated with hardware.
- OpenGL is based on this approach.

3D Polygons ➔ GRAPHICS PIPELINE ➔ Image Pixels
Choice of primitives

- Which primitives should an API contain?
  - small set - supported by hardware, or
  - lots of primitives - convenient for user
Choice of primitives

- Which primitives should an API contain?
  - small set - supported by hardware
  - lots of primitives - convenient for user
Choice of primitives

- Which primitives should an API contain?
  - small set - supported by hardware
  - lots of primitives - convenient for user

Performance is in 10s millions polygons/sec
portability, hardware support is key
Choice of primitives

• Which primitives should an API contain?
  ➡ small set - supported by hardware
  • lots of primitives - convenient for user

  GPUs are optimized for points, lines, and triangles
Choice of primitives

- Which primitives should an API contain?
  - small set - supported by hardware
  - lots of primitives - convenient for user

GPUs are optimized for points, lines, and triangles

Other geometric shapes will be built out of these
Two classes of primitives

Geometric: points, lines, polygons
Image: arrays of pixels
Point and line segment types

GL_POINTS

GL_LINES

GL_LINE_STRIP

GL_LINE_LOOP

[Angel and Shreiner]
Polygons

- Multi-sided planar element composed of edges and vertices.
- Vertices (singular: vertex) are represented by points.
- Edges connect vertices as line segments.
Valid polygons

- Simple
- Convex
- Flat
Valid polygons

- Simple
- Convex
- Flat
OpenGL polygons

- Only triangles are supported (in latest versions)

GL_POINTS  GL_TRIANGLES  GL_TRIANGLE_STRIP  GL_TRIANGLE_FAN
Other polygons

triangulation
Graphics Pipeline
Pipelining operations

An arithmetic pipeline that computes \( c + (a \times b) \)
Geometry: primitives – made of vertices
Vertex processing: coordinate transformations and color
Clipping and primitive assembly: output is a set of primitives
Rasterization: output is a set of fragments for each primitive
Fragment processing: update pixels in the frame buffer
Graphics Pipeline
(slides courtesy K. Fatahalian)
Vertex processing

Vertices are transformed into “screen space”
Vertex processing

Vertices are transformed into “screen space”

Vertices are transformed independently

Vertices

Vertices processor → Clipper and primitive assembler → Rasterizer → Fragment processor → Pixels
Primitive processing

Then organized into primitives that are clipped and culled…
Primitives are rasterized into “pixel fragments”
Rasterization

Primitives are rasterized into “pixel fragments”

EACH PRIMITIVE IS RASTERIZED INDEPENDENTLY
Fragment processing

Fragments are shaded to compute a color at each pixel
Fragment processing

Fragments are shaded to compute a color at each pixel

EACH FRAGMENT IS PROCESSED INDEPENDENTLY

Vertices → Vertex processor → Clipper and primitive assembler → Rasterizer → Fragment processor → Pixels
Pixel operations

Fragments are blended into the frame buffer at their pixel locations (z-buffer determines visibility)
Pipeline entities

Vertices

Primitives

Fragments

Fragments (shaded)

Pixels