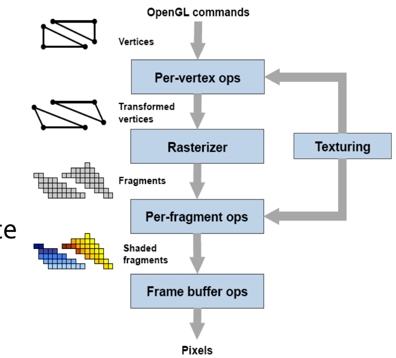
EECS 487: Interactive Computer Graphics

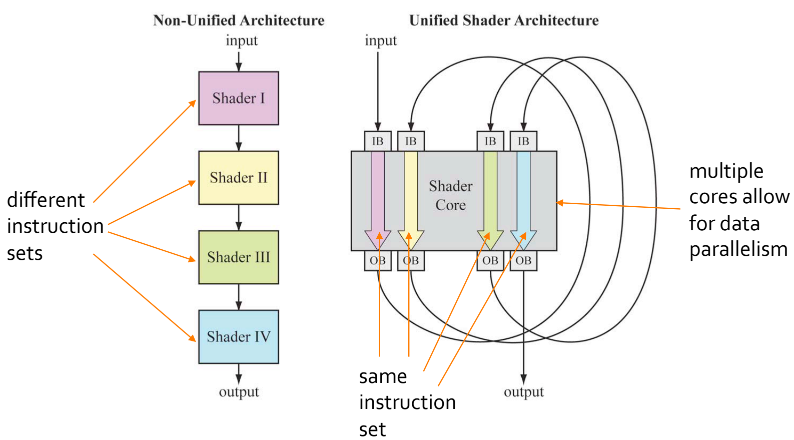
Lecture 4: GPU Overview
Raster Graphics
Line Rasterization

Pipeline Architecture

- Provides task parallelism, meaning?
- Benefits?
  - assume a 7-stage pipeline, each stage taking time *t* to complete, how long does it take to complete 7 jobs without pipelining? with pipelining?
- the output of a pipeline is determined by its bottleneck stage
- each stage can use parallel processing (data parallelism)

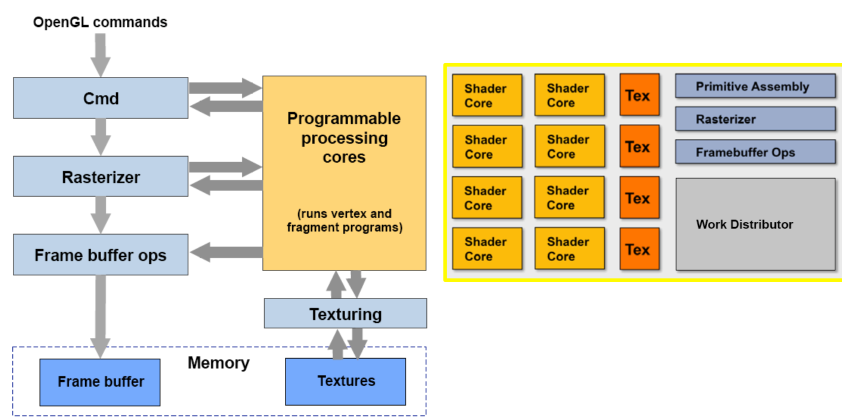


Unified Pipeline Architecture



[Akenine-Möller & Ström08]

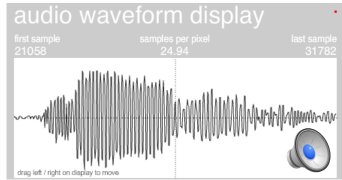
Multi-Core GPU with Unified Pipeline Architecture



[Hanrahan09]

Signal Digitization and Sampling

Analog audio signal:

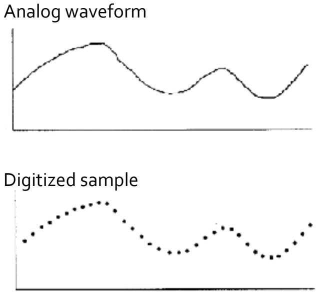


How to digitize the signal?

Sampling: reading the signal at certain rate to collect samples

Signal can be reconstructed from the samples

Analog waveform



Digitized sample

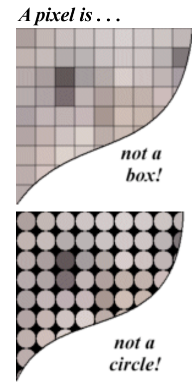
Analog waveform and digital equivalent

Raster Images

Pixel (picture element) ≔
- a discrete, point sample of a scene
- the digitized code values captured by an addressable photoelement (sensor hardware) [ISO]
  - including intensity, RGB color, depth, etc.

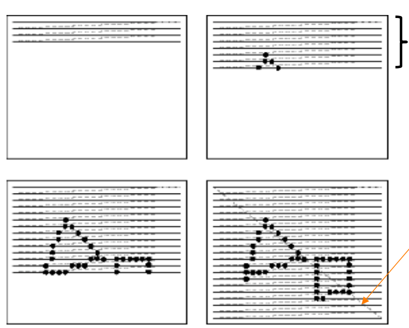
Image ≔ sampling of a scene rasterized as a 2D array of pixels

Raster ≔ a 2D array of pixels
- indexed (i, j)
- bottom-left pixel is $(0, 0)$
- top-right is (N_x-1, N_y-1)



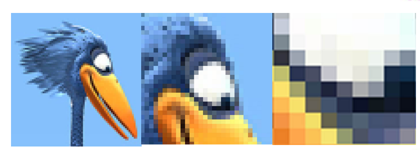
Raster Graphics Display

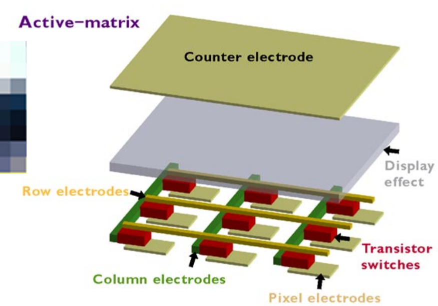
Generates and stores raster image in frame buffer
Reads frame buffer contents and turns on pixels
- requires constant update (60-80 Hz)
- most display devices nowadays are raster display (as opposed to vector display)



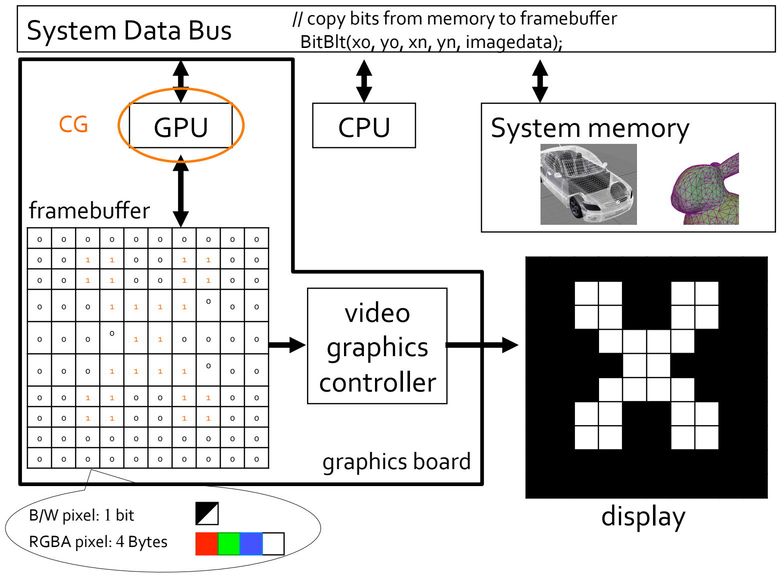
Example Raster Display

LCD:
- pixels turned on one row at a time via the row electrodes
  - individual pixels in the row turned on/off by appropriate voltages on the column electrodes
- active matrix pixels keep their state between updates (resulting in brighter display)





Raster Graphics Systems



Pixel Values

Three channels: Red, Green, Blue

- these three colors are enough to create a rich palette
- each channel takes a range of values
  - 24-bit color means
    - 3 bytes → one byte per channel
    - 0…255 for a color
- 1 byte per channel is enough for display
  - but may not be enough if you want to perform computations
    - instead, use floating point values 0.0 to 1.0 for computations

Be careful with your `float` to `int` conversion

Know when to use `floorf()`, `ceilf()`, and `rintf()`

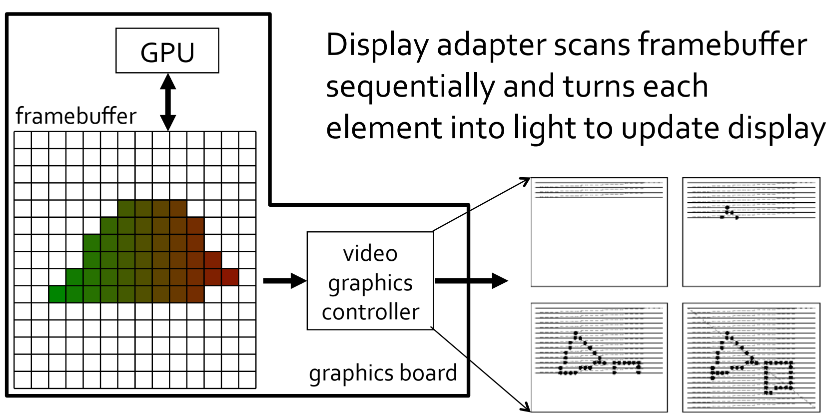
RGBA: A ($\alpha \in (0,1)$) is to control opacity

- useful for compositing: $\mathbf{c} = \alpha\, \mathbf{c}_{front} + (1-\alpha)\, \mathbf{c}_{back}$

Raster Graphics System

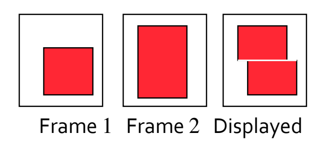
Framebuffer accessed by GPU randomly as each pixel is shaded

Display adapter scans framebuffer sequentially and turns each element into light to update display



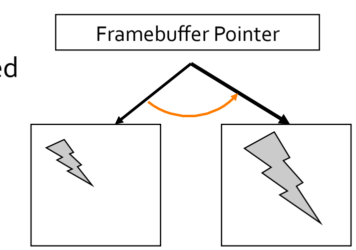
Double Buffering

Problem: image tearing if framebuffer is only partially updated when the next scan starts



Solution: double buffering:

- render to back buffer . . .
- . . . while the front buffer is displayed (multiple times if necessary)
- swap buffers during vertical blank/ retrace interval when back buffer is ready



Completing the Drawing

Issued GL commands may be stuck in buffers along the pipeline, e.g., waiting for more commands to be issued before sending them in batch

You need to flush all these buffers if you have no more commands to issue to effect execution start

- `void glFlush(void);`
  flushes the buffers and start execution of commands
- `void glFinish(void);`
  waits for commands to finish executing before returning
- `void glutSwapBuffers(void);`
  swaps back and front buffers if double buffering is in effect (as specified with `glutInitDisplayMode()`), implicitly calls `glFlush()`; no effect if single-buffered

Double Buffered GLUT Display Mode

```
#include <GL/glut.h>

int main(int argc, char *argv[])
{
  glutInit(&argc, argv);

  /* Create the window first before drawing! */
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
          /* double buffered, RGBA color */
  glutInitWindowSize((int)width, (int)height);
  wd = glutCreateWindow("Title");
              /* wd is the window handle */

  /* register callback functions/event handlers */
  glutReshapeFunc(reshape);
  glutDisplayFunc(display);

  glutMainLoop();
  return 0;
}
```

Double Buffered GLUT Display Mode

Skeletal `display()` callback for GL window to render a line:

```
void disp(void)
{
  /* Set color, linewidth etc */
  glBegin(GL_LINES);
    glVertex2f(x0,y0);
    glVertex2f(x1,y1);
  glEnd();
  glutSwapBuffers(); /* swap buffers */
  /* was glFlush() or glFinish();
     glutSwapBuffers() automatically calls glFlush() */
}
```

Rendering a Line

Lines are a basic primitive that must be done well

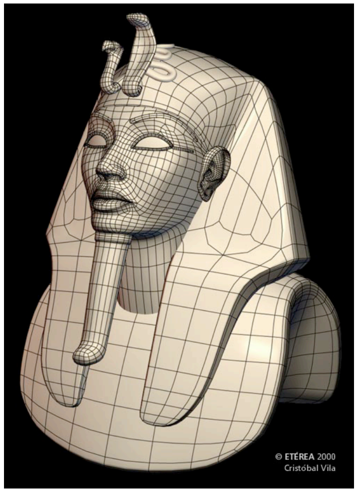
In OpenGL: just specify connection type and vertices:

```
glBegin(GL_LINES);
  glVertex2f(x0,y0);
  glVertex2f(x1,y1);
glEnd();
```

How is it implemented?

Requirements:

- continuous appearance, close to actual continuous line
- uniform thickness and brightness
- fast (line drawing happens a lot!)



Describing a Line and Line Segment

Three ways to describe a line:

- slope-intercept (or explicit) form: $y = mx + b$
- implicit* form: $f(x, y) = y - mx - b = 0$
- parametric form, using point and vector: $\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0)$

Given a line segment between (x_0, y_0) and (x_1, y_1), it can be described in slope-intercept form as $y = mx + b$, where

$$m = \frac{y_1 - y_0}{x_1 - x_0} \text{ and } b = y_0 - mx_0 = y_0 - \left(\frac{y_1 - y_0}{x_1 - x_0}\right)x_0$$

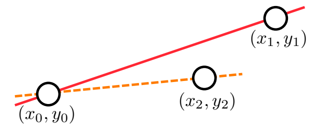
Or in implicit form: $f(x,y) = y - \left(\frac{y_1 - y_0}{x_1 - x_0}\right)x - (y_0 - mx_0) = 0$

* "implicit" means the equation doesn't generate points on the line, rather it confirms whether a point is on the line

Line Segment and Relative Position

A point at (x_2, y_2) is below the line segment if

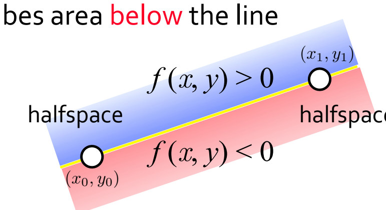
$$\frac{y_2 - y_0}{x_2 - x_0} < \frac{y_1 - y_0}{x_1 - x_0}$$



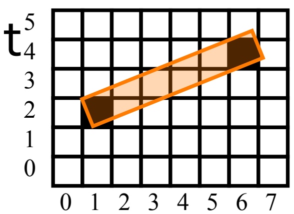
or, evaluating the line's implicit equation at (x_2, y_2) gives:

$$f(x_2, y_2) = y_2 - \left(\frac{y_1 - y_0}{x_1 - x_0}\right)x_2 - (y_0 - mx_0) < 0$$

In general, $f(x, y) > 0$ describes area above the line, and $f(x, y) < 0$ describes area below the line



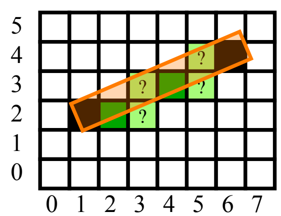
Drawing a Line Segment in Raster Graphics



To draw a line from $(1, 2)$ to $(6, 4)$

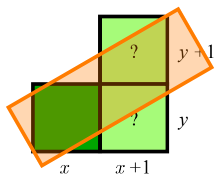
- for now assume integer coordinates

Want: thinnest line possible, with no gap, i.e., the pixels must be touching each other, even if only at the corner



Options:

- use the slope-intercept equation
- point sampling: turn on every pixel whose center the line "touches"
- Bresenham midpoint algorithm



Use Slope-Intercept

$$y = mx + b$$

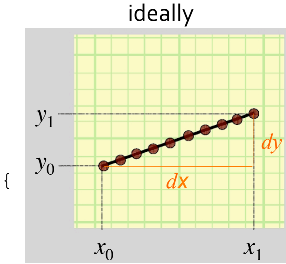
$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

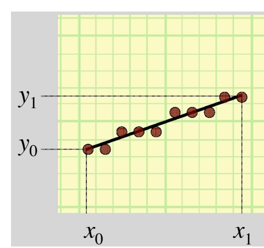
$$\Delta_y = m\Delta_x$$

$$b = y_0 - mx_0$$

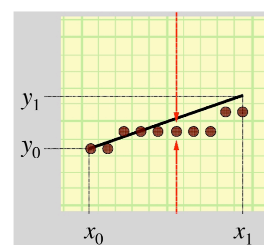
```
// step by dx = 1
// then dy is just = m
dy = (y1-y0)/(x1-x0);
y=y0;
for (x = x0; x <= x1; x++) {
  set(x, round(y));
  y += dy;
}
```

ideally

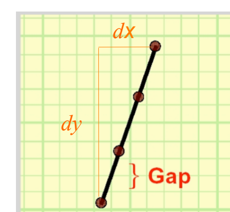




discretization rounding



error accumulation

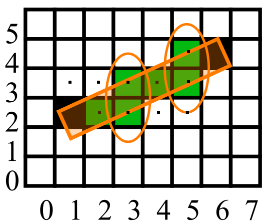


$|m| > 1$

O'Brien08

Point Sampling

Turn on every pixel whose center falls within the line

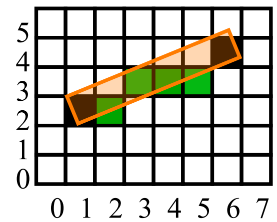


Problem:
not thinnest line possible

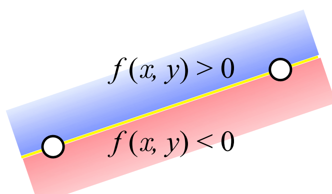
Consequence:

Midpoint Algorithm

Thinnest line possible, with no gap, i.e., the pixels must be touching each other, even if only at the corner



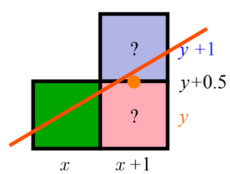
Midpoint Algorithm



Simple case first: assume $0 \le m \le 1$

Compute midpoint between the two pixels at $x+1$:

```
if (f(midpoint) < 0)
  line passes above midpoint,
  set pixel (x+1,y+1)
else
  line passes on or below midpoint,
  set pixel (x+1, y)
```



Implicit 2D Lines

In computing slope (m), to avoid dealing with fractional slope (or worse, zero denominator), restate $f(x, y)$ as:

$$f(x,y) = (y - mx - b)(x_1 - x_0) = 0$$

$$= \left(y - \left(\frac{y_1 - y_0}{x_1 - x_0} \right) x - \left(y_0 - \left(\frac{y_1 - y_0}{x_1 - x_0} \right) x_0 \right) \right) (x_1 - x_0)$$

$$= (x_1 - x_0)y - (y_1 - y_0)x - (x_1 - x_0)y_0 + (y_1 - y_0)x_0$$

$$= (x_1 - x_0)y - (y_1 - y_0)x + (x_0 y_1 - x_1 y_0)$$

Let: $C = x_0 y_1 - x_1 y_0 = \begin{vmatrix} x_0 & x_1 \\ y_0 & y_1 \end{vmatrix}$

Then $f(x, y) = \Delta_x y - \Delta_y x + C$ or,
for $A = (y_0 - y_1)$, $B = (x_1 - x_0)$: $f(x, y) = Ax + By + C$

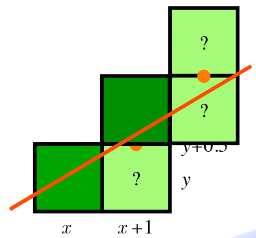
Incremental Midpoint Algorithm

Lines can then be computed incrementally (assume $0 \le m \le 1$):

$$f(x, y) = \Delta_x y - \Delta_y x + C$$

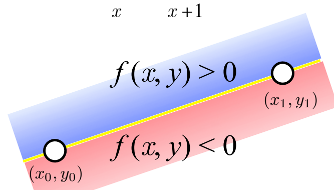
$$\begin{aligned} f(x+1, y) &= \Delta_x y - \Delta_y (x+1) + C \\ &= \Delta_x y - \Delta_y x - \Delta_y + C \\ &= f(x, y) - \Delta_y \end{aligned}$$

$$f(x+1, y+1) = f(x, y) + (\Delta_x - \Delta_y)$$



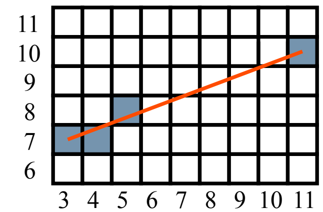
and drawn incrementally:

```
y = y0; dx = x1-x0; dy = y1-y0;
fmid = f(x0+1, y0+0.5);
for (x = x0; x <= x1; x++) {
    set_pixel(x, y);
    if (fmid < 0) {// line passes above midpoint
      y++; fmid += dx-dy;
    } else { fmid -= dy; }
}
```



Example: (3,7) to (11,10)

```
y = y0; dx = x1-x0; dy = y1-y0;
fmid = f(x0+1, y0+0.5);
for (x = x0; x <= x1; x++) {
  set_pixel(x, y);
  if (fmid < 0) { // line above midpoint
    y++; fmid += dx-dy;
  } else {
    fmid -= dy;
  }
}
```




```
dx = 11-3 = 8;
dy = 10-7 = 3;
dx-dy = 5;
fmid = f(4, 7.5) = 1
```

x	y	fmid
3	7	1
4	7	-2
5	8	3

Integer Only Line?

Can the line be drawn using only integers (no floats)?
to reduce round-off error and increase performance

$2f(x, y) = 0$ is a valid description of the line $f(x, y)$
hence, the algorithm can be rephrased as:

```
fmid2 = 2*f(x0+1, y0+0.5); // thus no more 0.5
dx2 = 2*(x1-x0); dy2 = 2*(y1-y0);
for (x = x0; x <= x1) {
  set_pixel(x, y);
  if (fmid2 < 0) { y++; fmid2 += dx2-dy2; }
  else fmid2 -= dy2;
}
```

Rasterization implemented in GPU as a fixed function

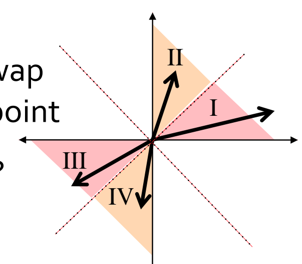
Be careful with your `float` to `int` conversion

Know when to use `floorf()`, `ceilf()`, and `rintf()`

What if $!(0 \le m \le 1)$?

Case I: $0 \le m \le 1$, done
Case II (steep slope): swap the x and y coordinates
Case III (going towards smaller x):
swap the two points
Case IV: swap the points and then swap
the x, y coordinates of each point

What to do in case of negative slope?

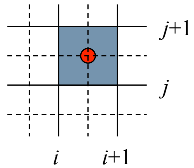


Midpoint algorithm can be used to draw circle and other conic sections (ellipses, parabolas, hyperbolas)

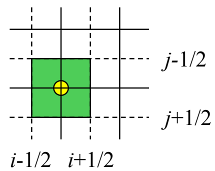
- exploit symmetry – only need to compute 1 octant

Image Coordinates Conventions

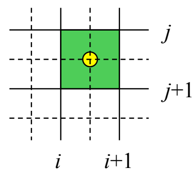
OpenGL (and Labs 1 & 2 & PA1)
- pixel center is at half-integers
- (0,0) at bottom left corner of screen (Cartesian coordinates)



Direct3D 9
- pixel center is at integers
- (0,0) at screen top-left corner (raster scan direction)



Direct3D 10/11
- pixel center is at half-integers
- but (0,0) at screen top-left corner



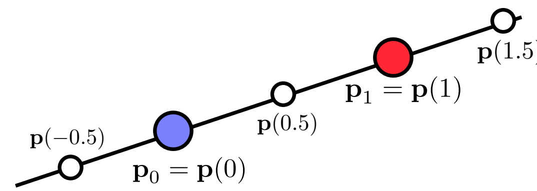
Parametric 2D Lines

Given two points $\mathbf{p}_0 = (x_0, y_0)$ and $\mathbf{p}_1 = (x_1, y_1)$, a line can be described as $\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} x_0 + t(x_1 - x_0) \\ y_0 + t(y_1 - y_0) \end{bmatrix}$

in parametric form: $\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0)$

or: $\mathbf{p}(t) = (1 - t)\mathbf{p}_0 + t\,\mathbf{p}_1$

or as a point $\mathbf{o}$ and a vector $\mathbf{d}$: $\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$



Linear, Affine, Convex Combinations

Given points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \ldots, \mathbf{p}_n$ and coefficients $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n$:

- linear combination:

  $\alpha_1\mathbf{p}_1 + \alpha_2\mathbf{p}_2 + \alpha_3\mathbf{p}_3 + \ldots + \alpha_n\mathbf{p}_n$
- affine combination:

  linear combination and $\alpha_1 + \alpha_2 + \alpha_3 + \ldots + \alpha_n = 1$
- convex combination:

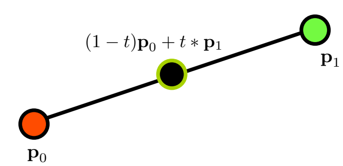
  affine combination and each $\alpha_i \geq 0$

What is described by this set of points in 3D?

$$\alpha\mathbf{p} + (1 - \alpha)\mathbf{q}, \text{ where } 0 \leq \alpha \leq 1$$

Parametric Line Drawing

$\mathbf{p}(t) = (1 - t)\mathbf{p}_0 + t\,\mathbf{p}_1$: is a convex combination (interpolation) of two points:
- any convex combination of two points lies on the straight line segment between the points

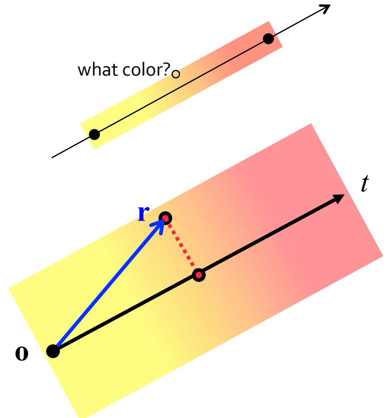


- a remarkably general concept, quite useful for blending
  - interpolation of: positions, colors, vectors

Coloring Thick Lines

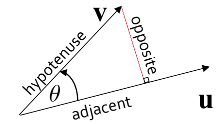
How to interpolate the color of each pixel on a thick (multi-pixel width) line?

1. Compute how far the pixel is from both endpoints
2. How to compute?
   - project pixel center onto the line to find parameter t of the parametric line equation
3. Correctly blended color can then be computed based on t



How to project pixel center onto the line?

Dot Product Review



The dot product of two vectors $\mathbf{u}$ and $\mathbf{v}$ is

a scalar value $\mathbf{u}\cdot\mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\|\cos\theta \;\Rightarrow\; \cos\theta = \dfrac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$

Another way to compute $\mathbf{u}\cdot\mathbf{v}$:

$$\mathbf{u}\cdot\mathbf{v} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} = \mathbf{u}^T\mathbf{v} = [u_0 \; u_1 \; u_2] \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} = \sum_{i=0}^{2} u_i v_i = u_0 v_0 + u_1 v_1 + u_2 v_2$$

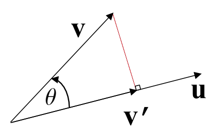
If $\theta = 90^0$ ($\mathbf{u} \perp \mathbf{v}$) then $\mathbf{u}\cdot\mathbf{v} = 0$

If $\mathbf{u}$ is normalized, i.e., $||\mathbf{u}|| = 1$, $\mathbf{u}\cdot\mathbf{u} = 1$

$\mathbf{u}\cdot\mathbf{v} = \mathbf{v}\cdot\mathbf{u}$ (commutative)

$(\mathbf{u} + \mathbf{v}) \cdot \mathbf{w} = \mathbf{u} \cdot \mathbf{w} + \mathbf{v} \cdot \mathbf{w}$ (distributive)

Dot Product Review



- Dot product can be used to project a vector orthogonally onto another vector:

$\mathbf{v}' = t\,\mathbf{u}$, $t = \dfrac{||\mathbf{v}'||}{||\mathbf{u}||}$, and recall that $\cos\theta = \dfrac{||\mathbf{v}'||}{||\mathbf{v}||}$

$$||\mathbf{v}'|| = ||\mathbf{v}||\cos\theta = ||\mathbf{v}||\frac{\mathbf{u}\cdot\mathbf{v}}{||\mathbf{u}||\,||\mathbf{v}||}$$

$$||\mathbf{v}'|| = \frac{\mathbf{u}\cdot\mathbf{v}}{||\mathbf{u}||}$$

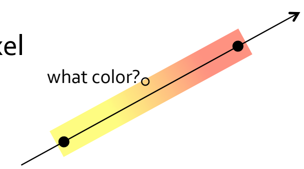
$$\mathbf{v}' = \left(\frac{\mathbf{u}\cdot\mathbf{v}}{||\mathbf{u}||^2}\right)\mathbf{u}$$

if $\mathbf{u}$ is normalized,

$\mathbf{v}' = (\mathbf{u}\cdot\mathbf{v})\mathbf{u}$

Coloring Thick Lines

How to interpolate the color of each pixel on a thick (multi-pixel width) line?



Need to know how far the pixel is from both endpoints (How?)

1. Project pixel center onto the line to find parameter t of the parametric line equation
   - dot product with a unit length vector (e) == performs projection onto the unit length vector
2. Correctly blended color can then be computed based on $t = ||(\mathbf{r}\cdot\mathbf{e})\mathbf{e}|| = \mathbf{r}\cdot\mathbf{e}$