

COMS 30115

Rasterisation

Carl Henrik Ek - carlhenrik.ek@bristol.ac.uk

February 25th, 2019

<http://www.carlhenrik.com>

- Shadows
- How to think about optimisations in raytracing?
 - Datastructures
 - Instruction level
 - Fidelity level
- Summarisation of Raytracing

Today

- Start with Rasterisation
- Image space rendering
- Line drawings

sadly the books is rather empty on the material in this lecture

- Bresenham [URL](#)
- Paper on line drawing algorithms [URL](#)

Real-time Graphics¹



¹Ghost Recon Wildlands

Real-time Graphics¹

- Remember the generative model?

$$I = f(x)$$

- how much data is actually I?
- 50 HD images/second

$$50 \times 1920 \times 1080 \times 4 \approx 400mb/s$$

- Interactive \Rightarrow Latency important
- Minimum and consistent performance important



- Efficient implementations
 - don't think how does physics work, think of the effect physics has
 - emulate the effect in a simpler way
- Adapt to hardware
 - write code that respects the hardware

C/C++

- Available on nearly all machines

²Mikael Kalms, (DICE, 1998-2016, Fall Damage 2016-)

- Available on nearly all machines
- Can access native APIs on nearly all OSes

²Mikael Kalms, (DICE, 1998-2016, Fall Damage 2016-)

- Available on nearly all machines
- Can access native APIs on nearly all OSes
- Control over memory management

²Mikael Kalms, (DICE, 1998-2016, Fall Damage 2016-)

- Available on nearly all machines
- Can access native APIs on nearly all OSes
- Control over memory management
- Control over memory layout

²Mikael Kalms, (DICE, 1998-2016, Fall Damage 2016-)

- Available on nearly all machines
- Can access native APIs on nearly all OSes
- Control over memory management
- Control over memory layout
- No garbage collection

²Mikael Kalms, (DICE, 1998-2016, Fall Damage 2016-)

- Available on nearly all machines
- Can access native APIs on nearly all OSes
- Control over memory management
- Control over memory layout
- No garbage collection
- *"the most advanced assembler on the market"*

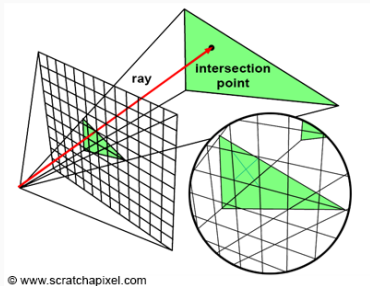
²Mikael Kalms, (DICE, 1998-2016, Fall Damage 2016-)

Code

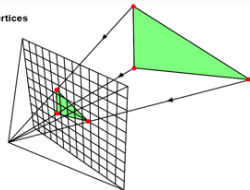
```
gcc -g -o tst tst.c  
objdump -d tst > tst.s
```

Rasterisation

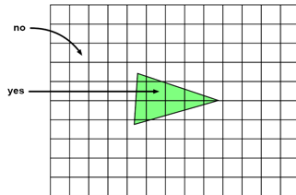
Part I vs Part II



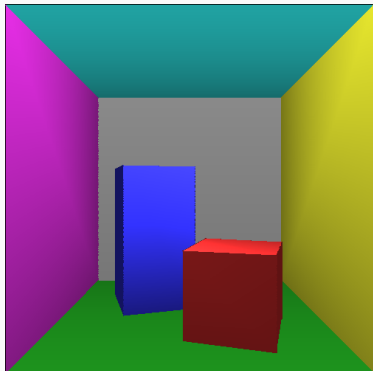
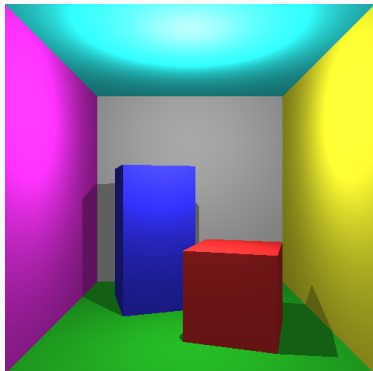
1) Project vertices



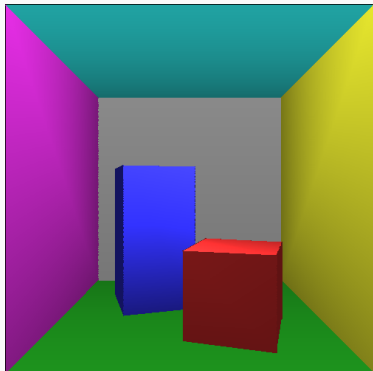
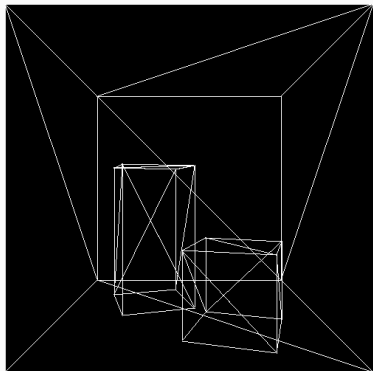
2) Loop over pixels. Does the pixel lie in the triangle?



Part I vs Part II



General Concept

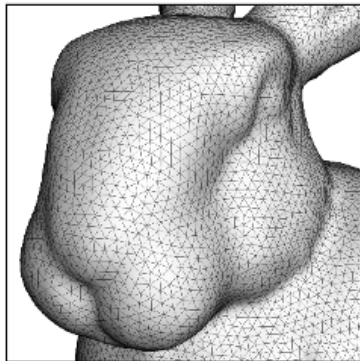
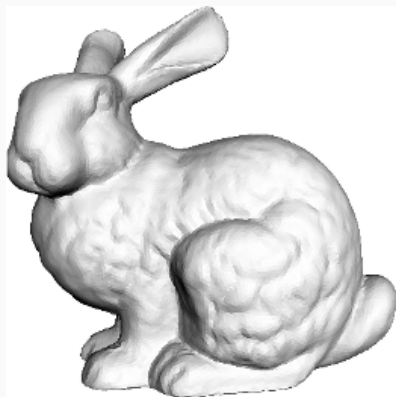


- Raytracer simple and slow
 - Calculate shading in the "world"
 - *What part of the world matches each pixel?*
 - Calculations in world

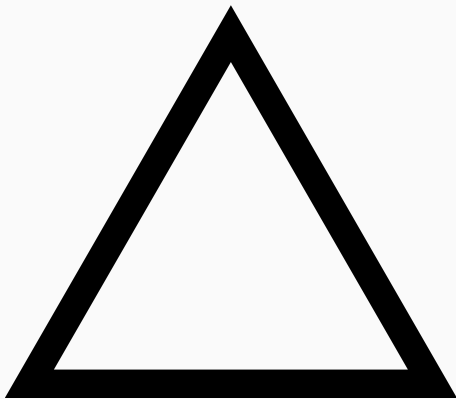
- Raytracer simple and slow
 - Calculate shading in the "world"
 - *What part of the world matches each pixel?*
 - Calculations in world
- Rasteriser messy but fast
 - Calculate shading in the image
 - *What pixel does this part of the world match to?*
 - Calculations in screen space

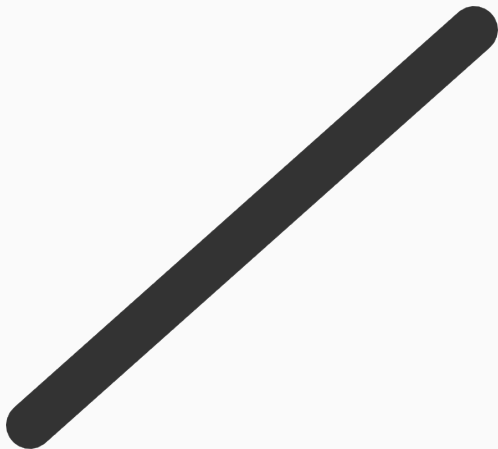
Structure of Part II

1. How to draw in screen space
 - primitives (lines, triangles)
 - how to draw discrete data
2. How to do sparse computations
 - interpolation
3. How to solve visibility problem
4. Shading in image space (vertex shading)
5. Mappings (Texture etc.)



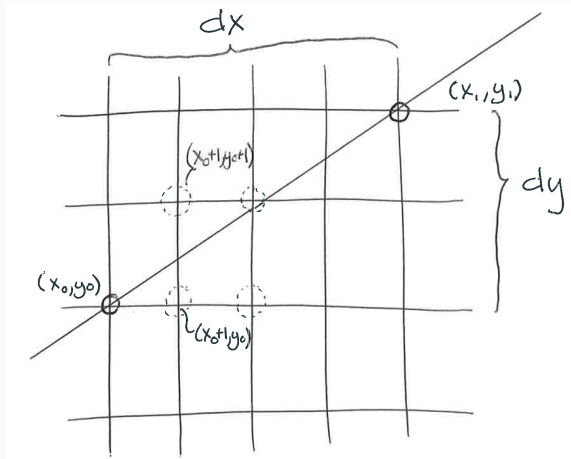
Triangles





Line Drawing

Digital Differential Analyser (DDA)



$$y = k \cdot x + m$$

Digital Differential Analyser (DDA)

Code

```
putpixel(x,y);  
dx = x1-x0;  
dy = y1-y0;  
if(dx>dy){steps = abs(dx);}   
else{steps = abs(dy);}   
xp = dx/steps; yp=dy/steps;  
for(int i=0;i<steps;i++)  
{  
    x += xp;  
    y += yp;  
    putpixel(round(x),round(y));  
}
```

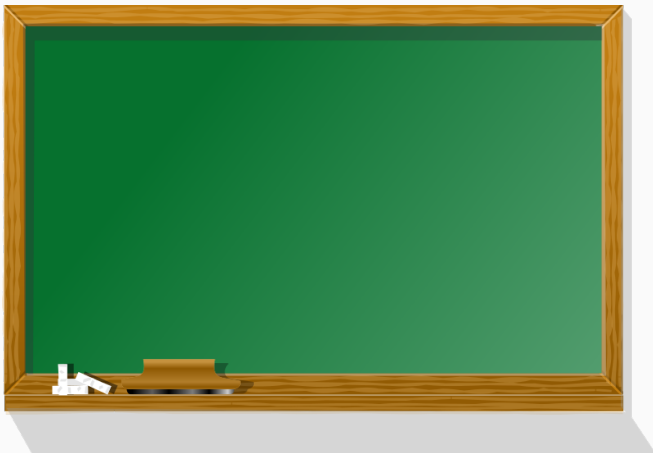
Drawing lines like a boss

- DDA algorithm is slow
 - rounding of floats
 - float additions

Drawing lines like a boss

- DDA algorithm is slow
 - rounding of floats
 - float additions
- Optimise by reducing to more cases that are less general

Bresenham's Derivation



Bresenham's Line Drawing

Code

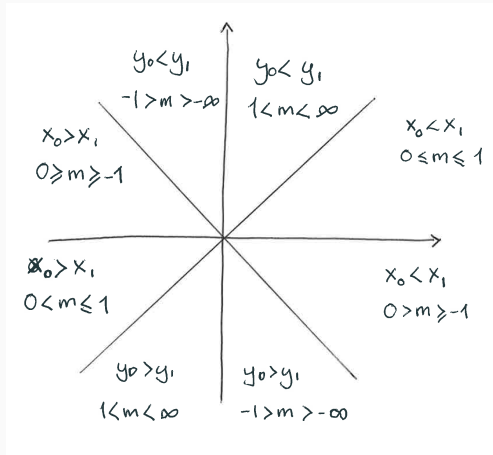
```
putpixel(x0,y0);  
dx = x1-x0;dy=y1-y0;  
2dx = 2*dx; 2dy = 2*dy;  
2dydx = 2dy - 2dx;  
d = 2dy-dx;
```


Bresenham's Line Drawing

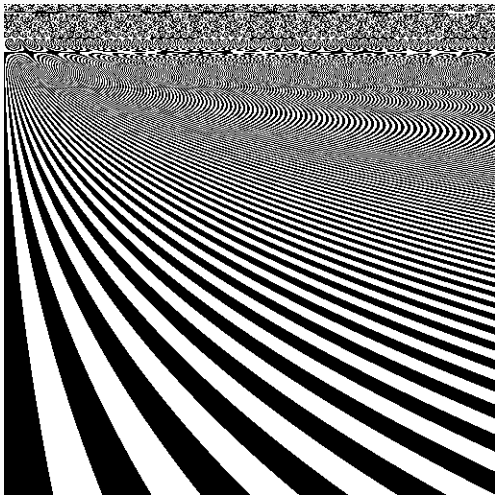
Code

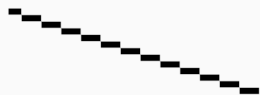
```
for(int i=0;i<dx;i++){  
    if(d<0){  
        x += 1;  
        d += 2dy;  
    }  
    else{  
        x+=1;  
        y+=1;  
        d += 2dydx;  
    }  
    putpixel(x,y);  
}
```

Bresenham's Line Drawing



- Completely integer
- 8 (4) cases for lines (really 4+4)

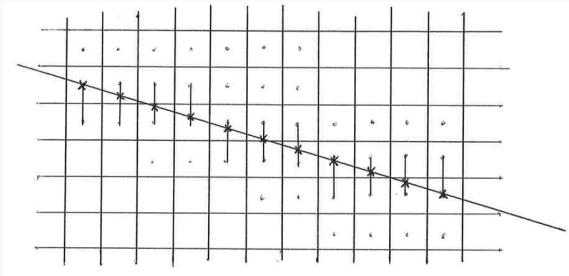




Normal line

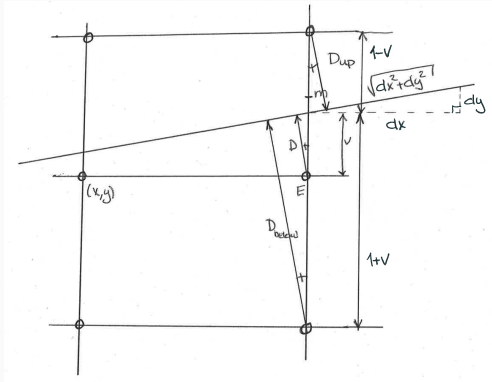


Anti-aliased line



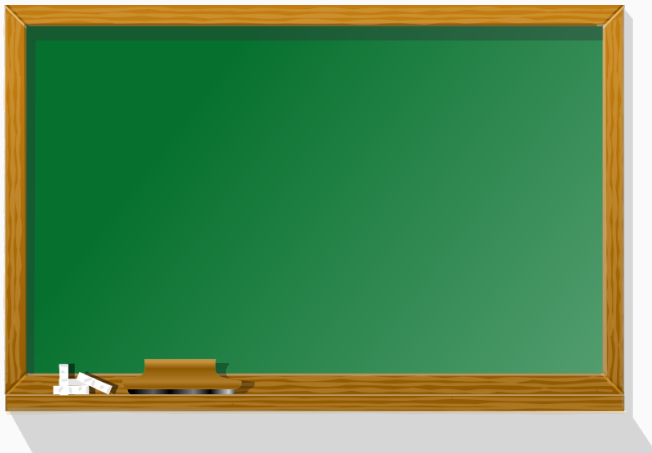
- Compute distance between line and pixel
- Set pixel above and below
- Intensity sum to 1

³https://en.wikipedia.org/wiki/Xiaolin_Wu%27s_line_algorithm



- Area that pixel covers important
- Weight pixels with perpendicular distance

Gupta-Sproull: derivation



Code

```
//compute constants A,B  
//1. Run Bresenham and get d  
if(d<0) //E pixel  
{  
    D = A*(d+dx);  
    Dup = B-D;;  
    Dbelow = B+D;  
    // look-up shading based on D
```


Summary

- Work in image space
- Make sparse computations and interpolate
- Example of interpolation: Lines
- Graphics is expensive, if we want realtime we have to think
- Optimise by making many simpler cases
 - cheap especially for simple conditions on integers

Lecture next primitive

- Triangles
- Perspective correct interpolation

Lab start with Lab 2

eof