OpenGL/GLUT
Double Buffering
Scan Conversion Algorithm
Clipping

CS3241 Computer Graphics

Why OpengGL?

An Email from a Student

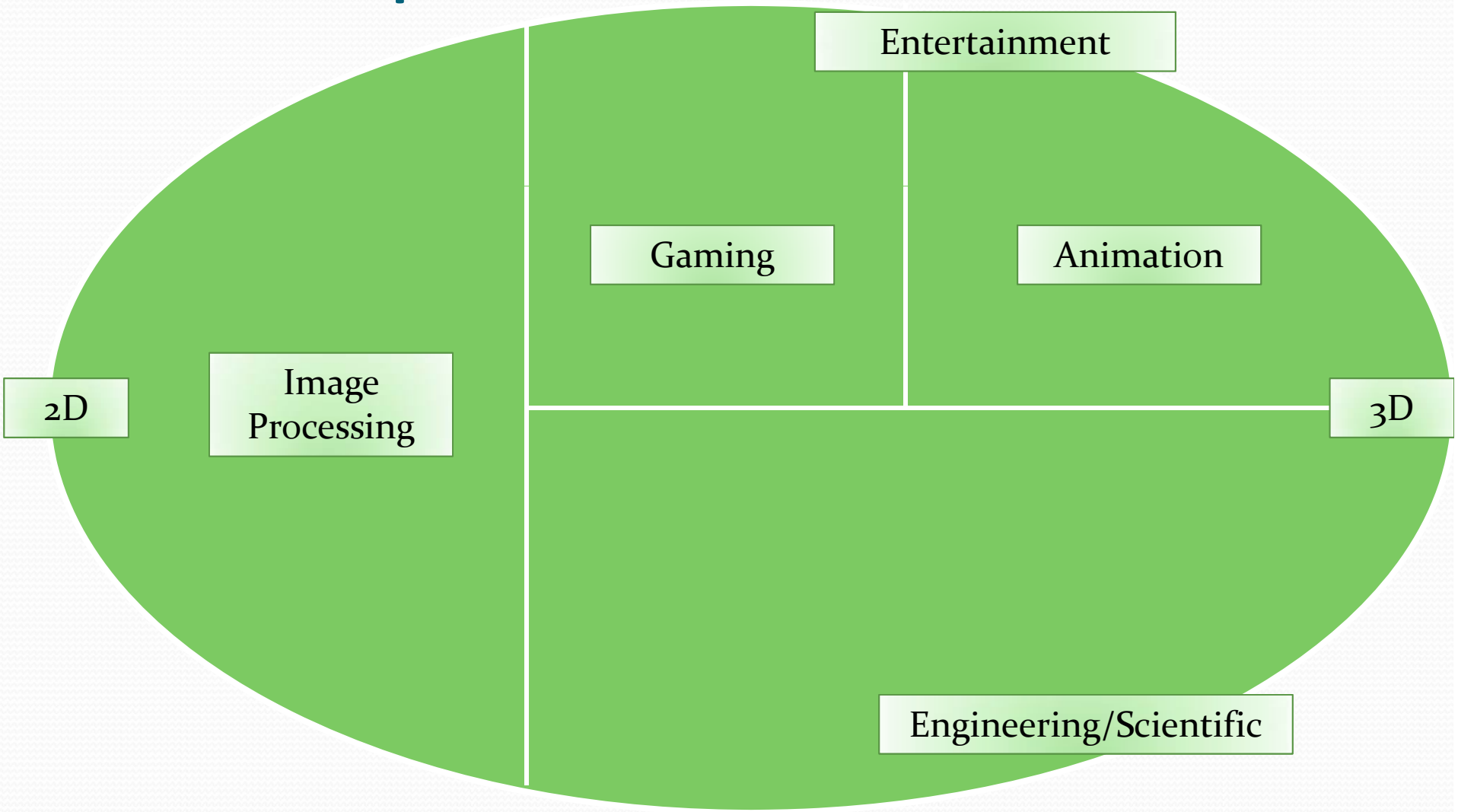
I've been trying out OpenGL after this week's lecture and I actually do have fun playing around with the functions. However, one question remains stuck at the back of my head, "Why exactly do we learn OpenGL for?" I still do not understand why we need to learn OpenGL when there are tools like Adobe Photoshop, Illustrator, After Effects, etc to help us with drawing graphics? Do not get me wrong, I'm not saying that I find learning OpenGL is a waste of time. I just do not understand how it relate to us.

Are we learning OpenGL just so we can write graphical programs? Rather than using it to really draw graphics?

Thanks for clarifying.

The Graphic World

The Graphic World



“Graphic” Software

- Image processing
  - Mainly 2D images
  - Adobe Photoshop, Illustrator, After Effects, Instagram, etc.
- 3D software
  - CAD/CAM, Engineering/Scientific computations
  - Animation software
    - E.g. 3DS Max, Alas Wavefront
- Game Engines

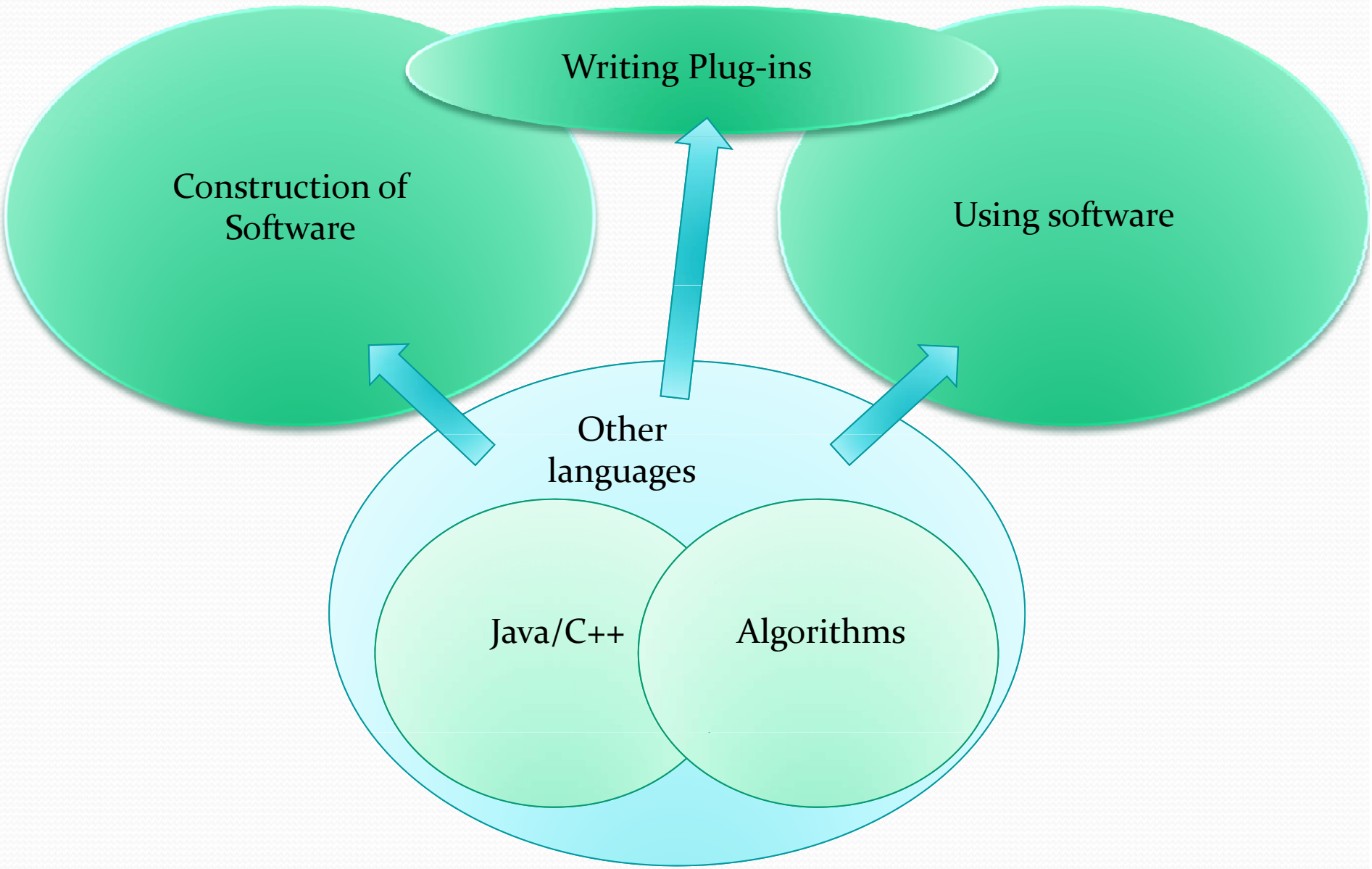
The Role of OpenGL & G. Algo

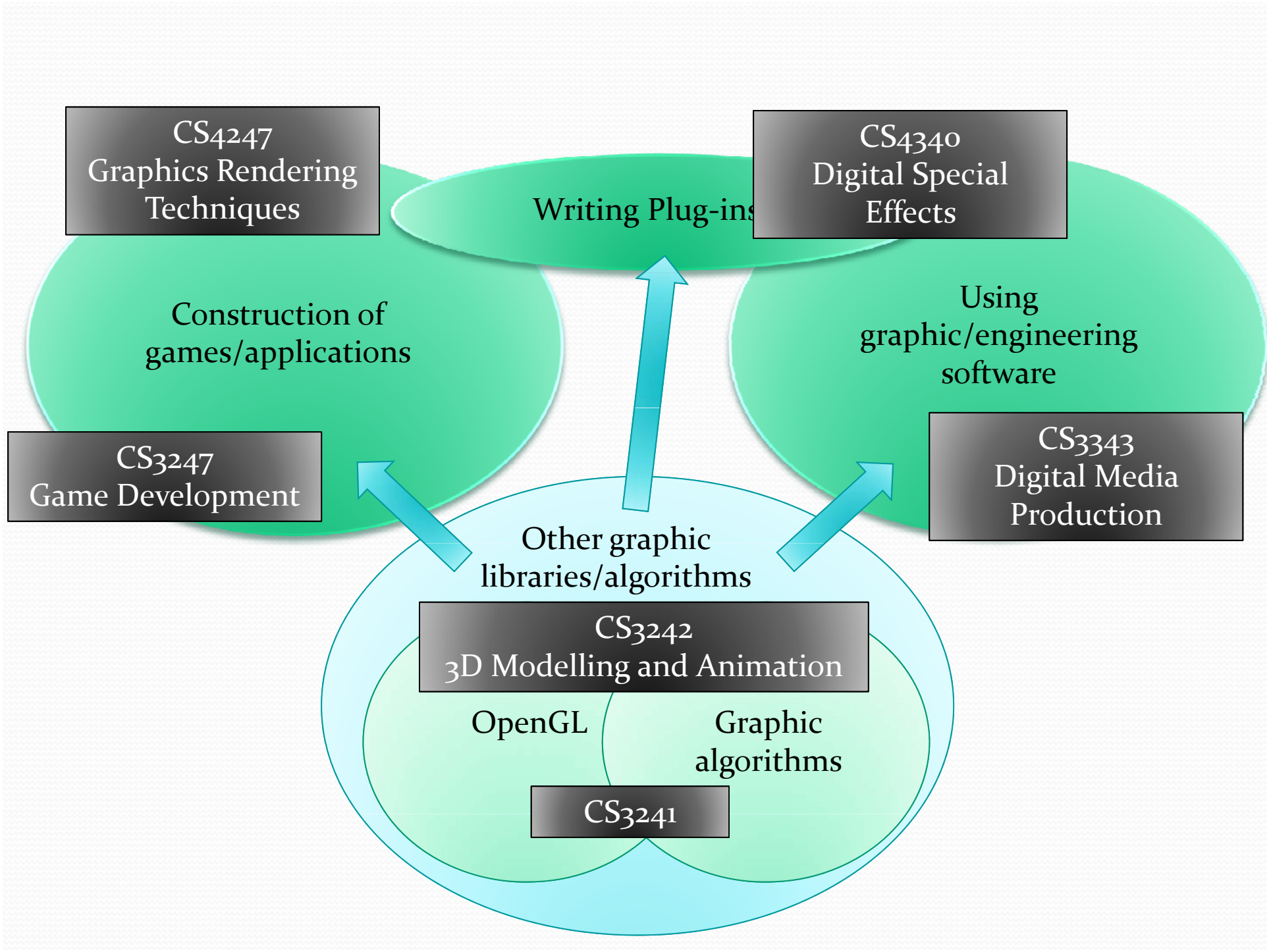
Graphic Software

- Understand how to draw a Nyan Cat in 3D
  - Namely, algorithms behind it, e.g. transformation, textures, etc
- Need a tool to learn
  - E.g. OpenGL
  - In which, you should be able to extend the ideas to other graphic API

MS Excel

- Understand how to sort a bunch of data
  - Namely, algorithms behind it, e.g. sorting
- Need a tool to learn
  - E.g. Java, C++, Scheme, etc.
  - In which, you should be able to extend the ideas to other languages


Writing Plug-ins
Construction of Software
Using software
Other languages
Java/C++
Algorithms


CS4247
Graphics Rendering
Techniques
Writing Plug-ins
CS4340
Digital Special
Effects
Construction of
games/applications
Using
graphic/engineering
software
CS3247
Game Development
CS3343
Digital Media
Production
Other graphic
libraries/algorithms
CS3242
3D Modelling and Animation
OpenGL
Graphic
algorithms
CS3241

What can we get out of OpenGL?

- Understanding graphic algorithms and experimenting them
  - In order to use future graphic software effectively
- Program your own graphic software
  - E.g. Game programming, artistic/scientific programming
- Create your own animation
- Understand the bugs of current programs, apps or animations, etc
- Research

An Overview of OpenGL/GLUT Structure

An OpenGL/GLUT program is like…

Programmer

OpenGL

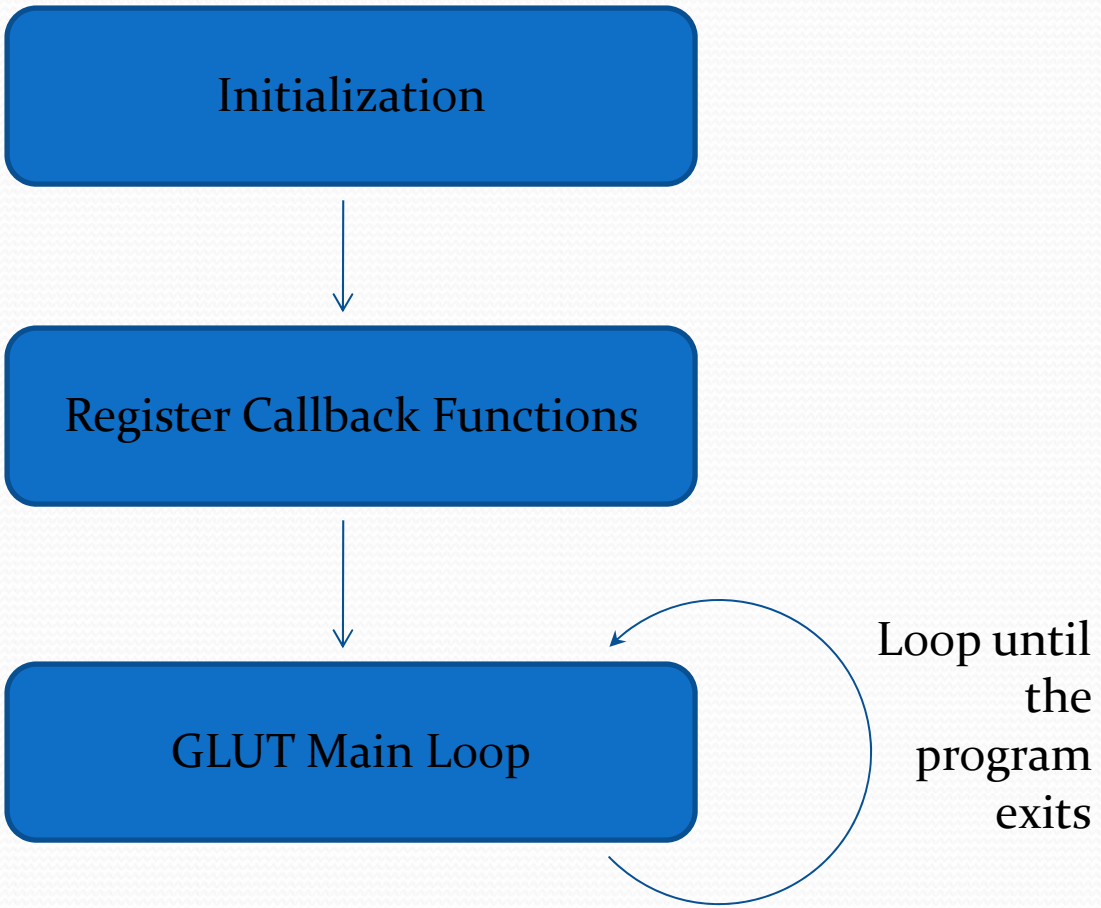
GLUT

Overall Program Flow (GLUT)

Initialize user variables, load geometry, set window size, Open window with attributes (e.g. RGB, double-buffer, z-buffer, etc. .)

Register user define “callback” functions, e.g. reshape, keyboard, mouse, idle . . .

`glutMainLoop()`



```
int main()
{
    glutCreateWindow("CS3241: Hello OpenGL!");
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}
```

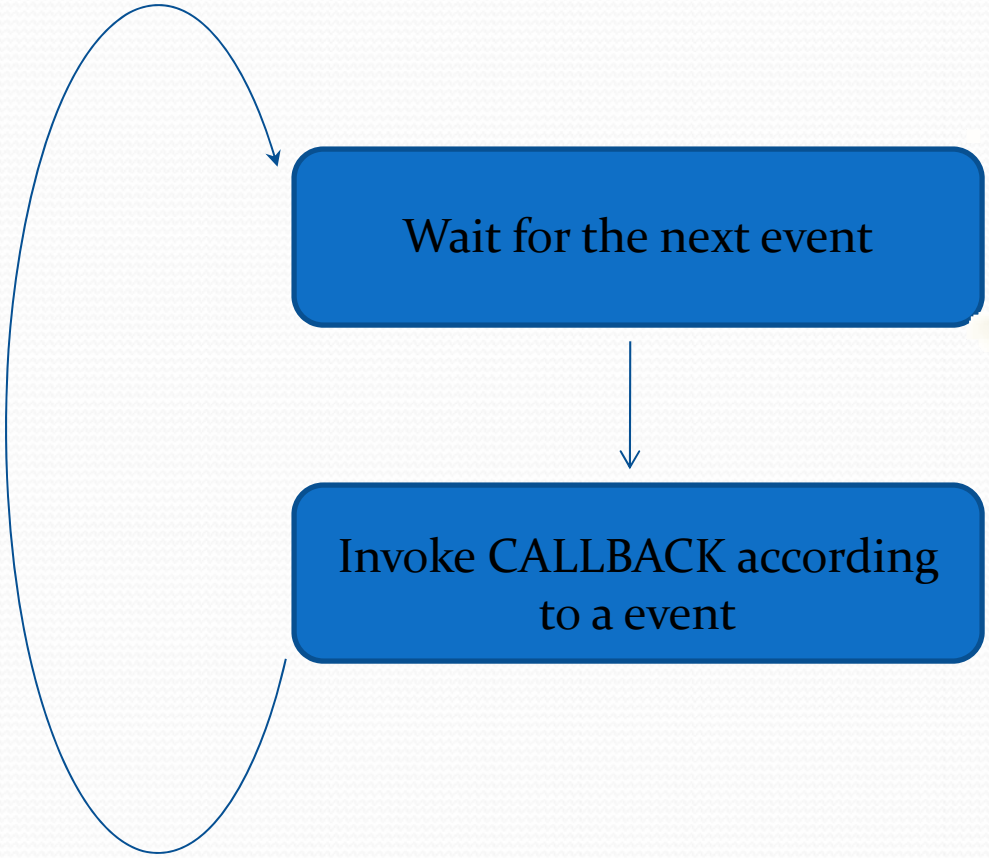
Initialization

GLUT Main Loop

Register Callback Functions

So if my program receives an event that it needs to display its window, I will call the function "`mydisplay()`"

GLUT Main Loop

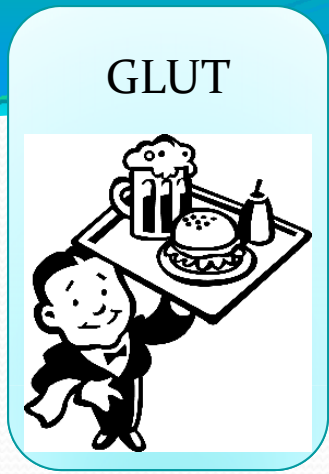


(queue up events if there are more than one)

Events

- Just like customer (user) orders
- Event driven
  - Meaning if there is no event, the program will not do anything (namely, entering the idle state)
- MS Windows, or (whatever X-win) will send an event to your program if anything happens
  - E.g. exposure of your window, movements of the keyboard, mouse, or space-ball, etc.
- Depends on the callback functions registered previously, GLUT will call the according functions
  - E.g. when I receive a display event, the program will call the function registered (e.g. `mydisplay()`)

Some Popular Event Types



- Display events
  - The display callback is executed whenever GLUT determines that the window should be refreshed, for example,
    - When the window is first created
    - When a window is exposed
    - When the user program changes the display contents
    - Etc…
  - Register a function by
    - `glutDisplayFunc(mydisplay);`
- Keyboard events
  - `glutKeyboardFunc(keyboard);`
- Mouse events
  - `glutMouseFunc(mouse);`
- And more, e.g. spaceball, reshape, etc.

OpenGL Functions Types

- **Drawing Geometry Primitives**
  - Points, line segments, polygons, etc.
- **Changing/querying Attributes/states**
  - Drawing/background colors, line width, shading modes, etc.
- **Manipulating Transformations**
  - Viewing
  - Modeling
- **Control** (GLUT): setup application
- **Input** (GLUT): handle events

OpenGL States/Attributes Examples

- Current drawing/background color
- Size and width (points, lines)
- Stipple pattern (lines, polygons)
- Polygon mode
  - Display as filled: solid color or stipple pattern
  - Display edges
  - Display vertices
- Current transformation matrices
- Various matrix stacks
- Modes of doing things,
  - e.g. shading mode
    - Flat vs smooth
- Single or double buffers
- Etc. etc….

Lighting Model (Flat)

Lighting Model (Smooth)

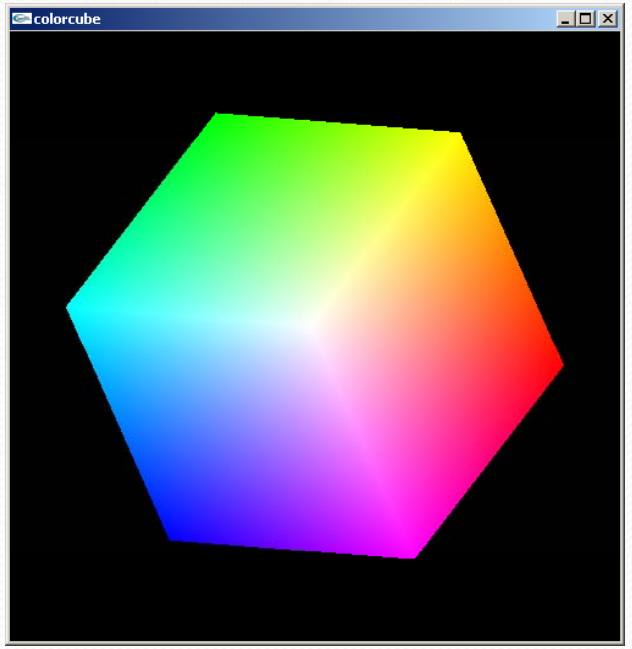
Smooth Color

- Default shading mode is *smooth* shading
  - OpenGL interpolates vertex colors within a polygon
- Alternative shading is *flat shading*
  - Color of first vertex determines fill color
- Functions to switch between the two:

  `glShadeModel(GL_SMOOTH)`

  or

  `glShadeModel(GL_FLAT)`



A More Complete Example

```
#include <GL/glut.h>
...
int main(int argc, char** argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(500,500);
  glutCreateWindow("simple");
  glutDisplayFunc(mydisplay);

  glClearColor (0.0, 0.0, 0.0, 1.0);
  glColor3f(1.0, 1.0, 1.0);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluOrtho2D(-1.0, 1.0, -1.0, 1.0);

  glutMainLoop();
}
```

glut.h includes **gl.h**

Not necessary unless you are using "X-widows" on Unix.

Defines app properties

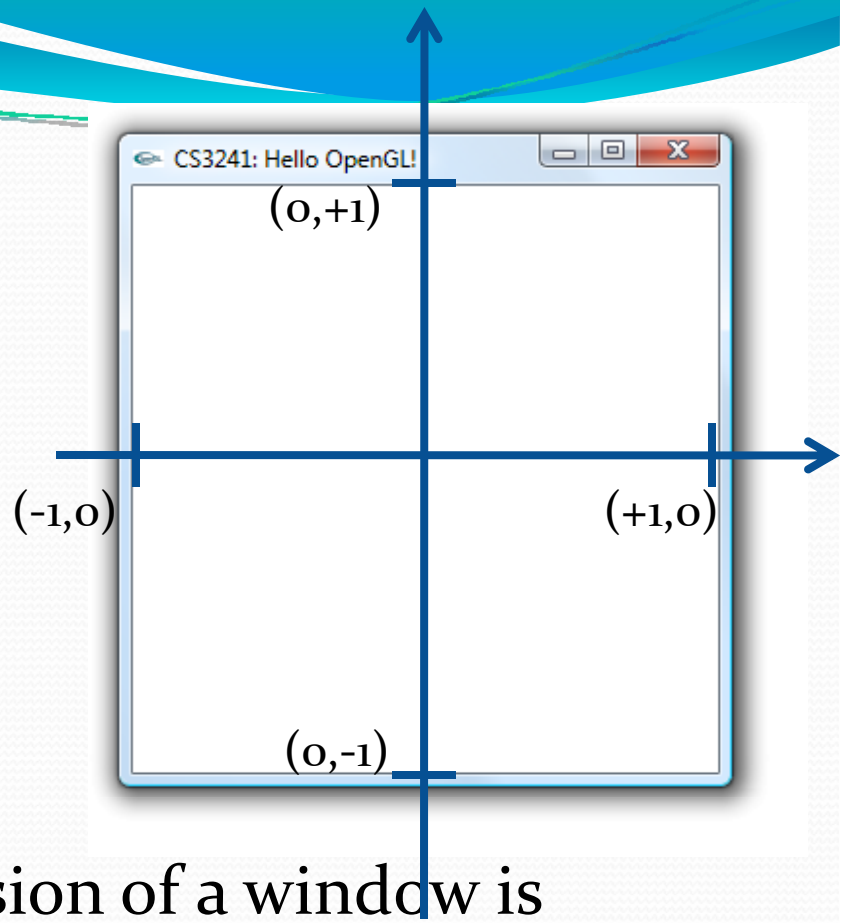
define window properties

display callback

viewing volume

enter event loop

Screen Dimension Setting (Viewing Volume)

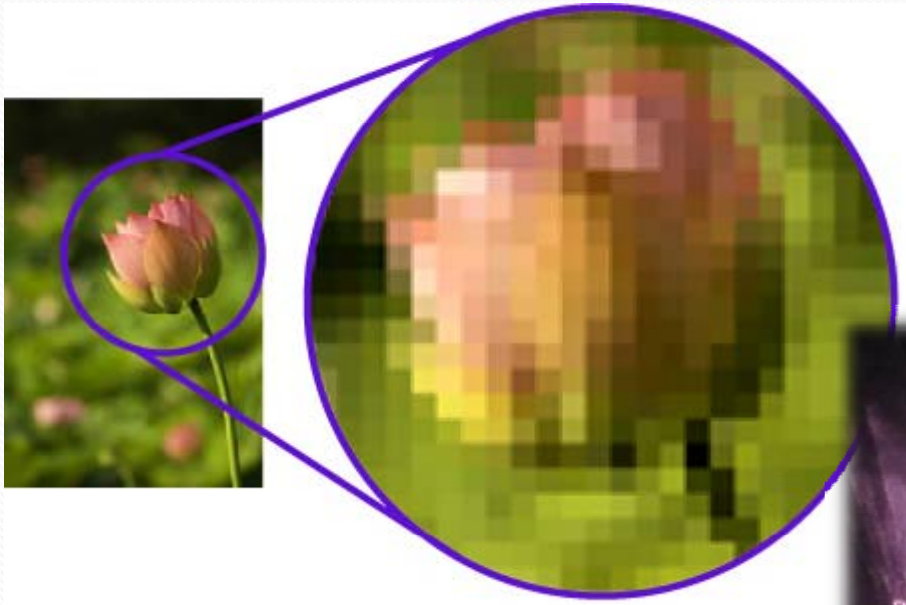


- Remember the default dimension of a window is [-1,1]x[-1,1]
- We can change it by the function:

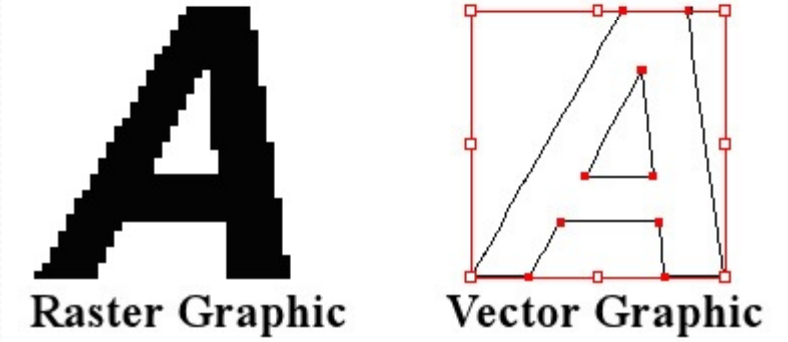
```
gluOrtho2D(left, right, bottom, top)
```

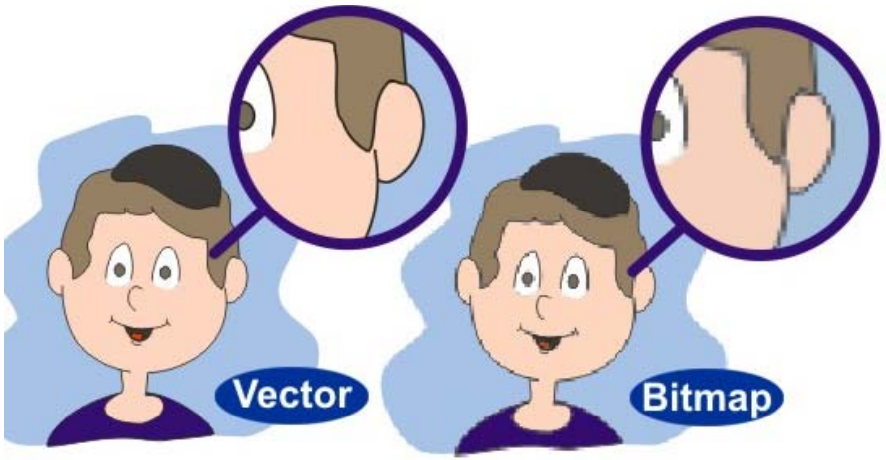
Screen Buffer (Memory)

Rasterization vs Vector Graphics



Rasterization vs Vector Graphics





Screen Buffers

- Buffer = memory
- Screen buffer
  - Array of memory that stores the values of a pixel
  - The values of a pixel include
    - Color, e.g. RGB
    - Degree of transparency, namely, the alpha (A) in RGBA (Next lab session)
    - The depth in z direction (Lecture 6)
    - Etc….

Double Buffering*

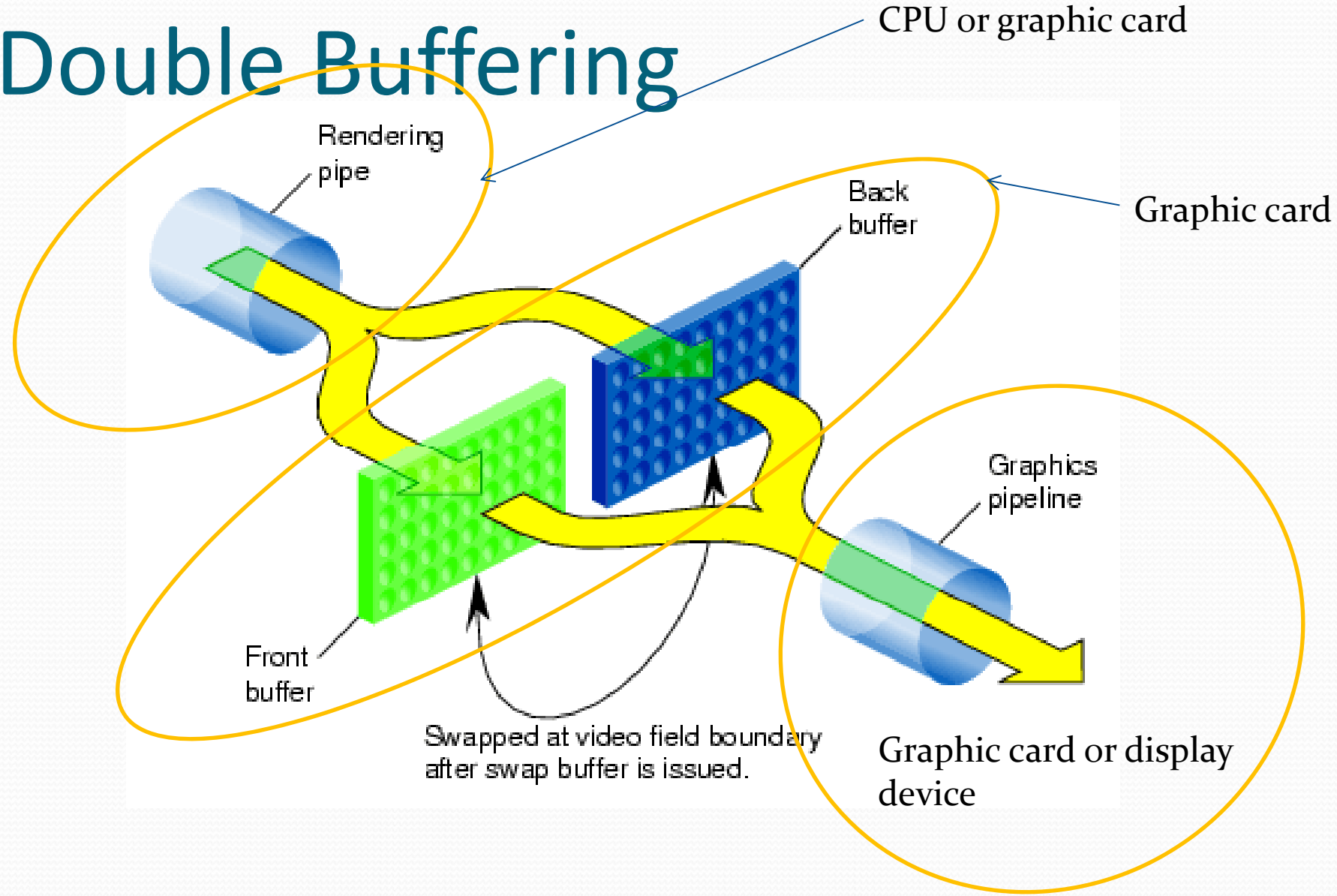
- What if you need 5 seconds to draw a picture…
- Instead of one color buffer (memeory), we use two
  - **Front Buffer**: one that is displayed but not written to
  - **Back Buffer**: one that is written to but not displayed
- In **`main.cpp`**
  - **`glutInitDisplayMode(GL_RGB | GL_DOUBLE)`** ← Setup Double buffer mode before you create your window.
  - At the end of the display callback buffers are swapped

```
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT|….)
    ...
    /* draw graphics here */
    ...
    glutSwapBuffers();
}
```

← Call swaping buffer after a page of picture is finished. What if you forget this? No picture on the screen!

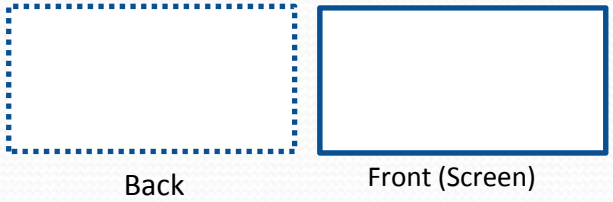
*The truth is we **always** use double-buffering, single buffer graphics apps are rare!

Double Buffering

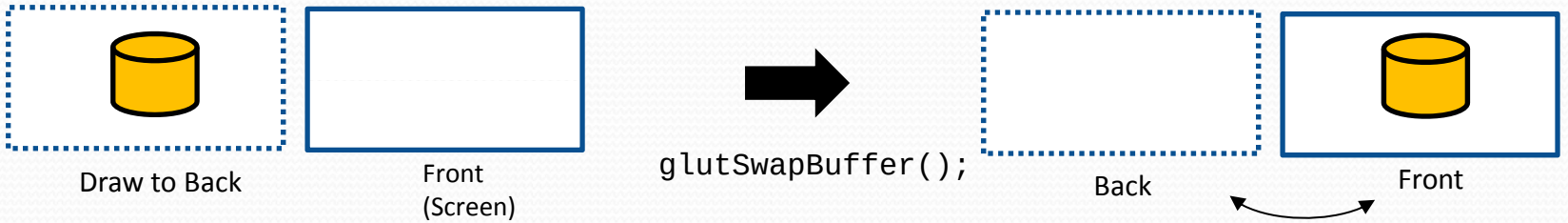


Double Buffering (use two buffers)

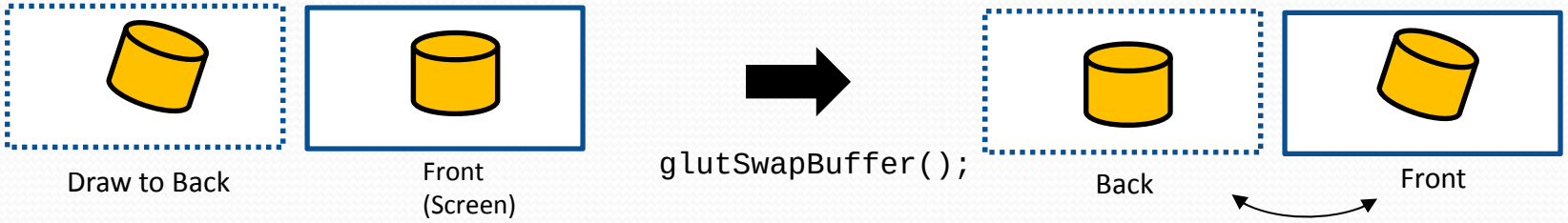
1- App opens, both buffers are empty



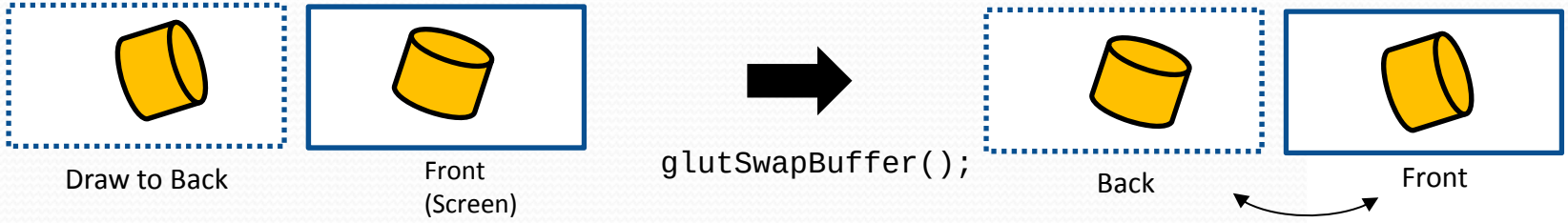
2- Draw some awesome graphics in the BACK, then swap buffers



3- Draw some more awesome graphics in the BACK, then swap buffers



4- Draw even more awesome graphics in the BACK, then swap buffers [REPEAT]



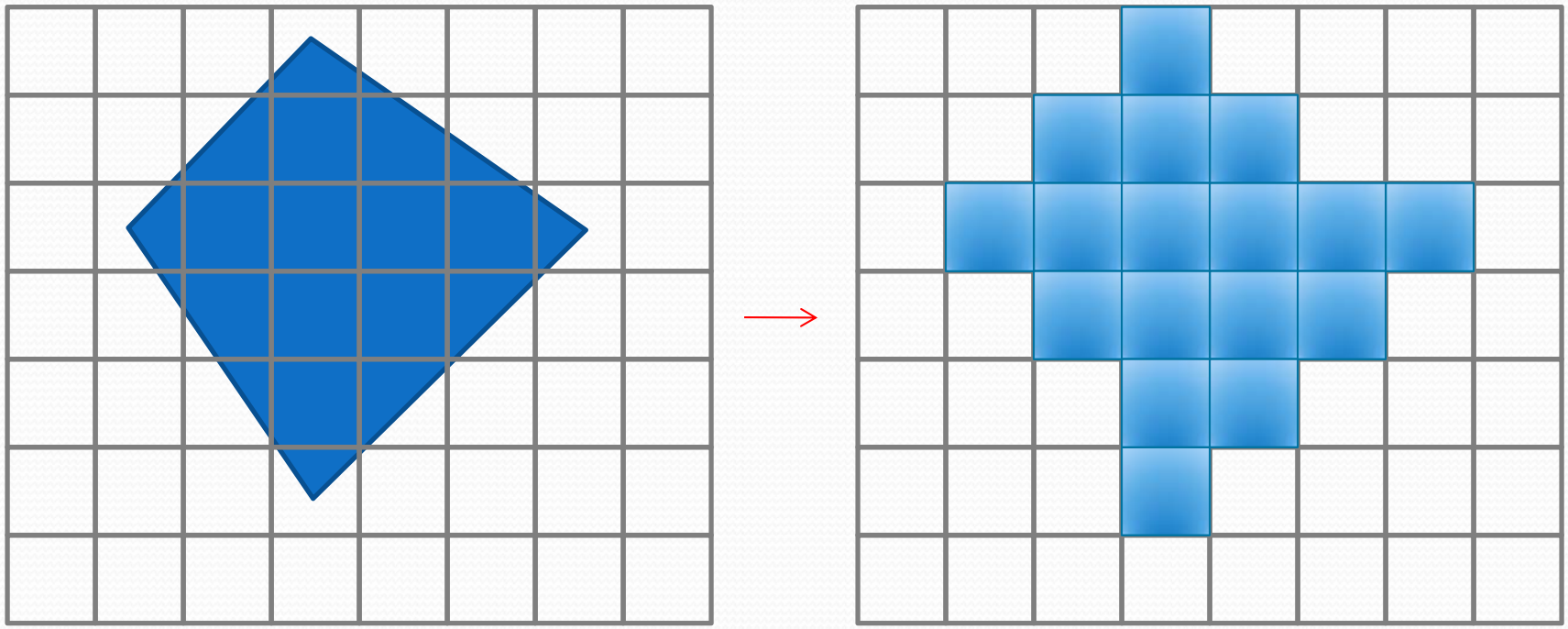
Scanline Conversion Algorithm

(SCA)

Scan Conversion Algorithm

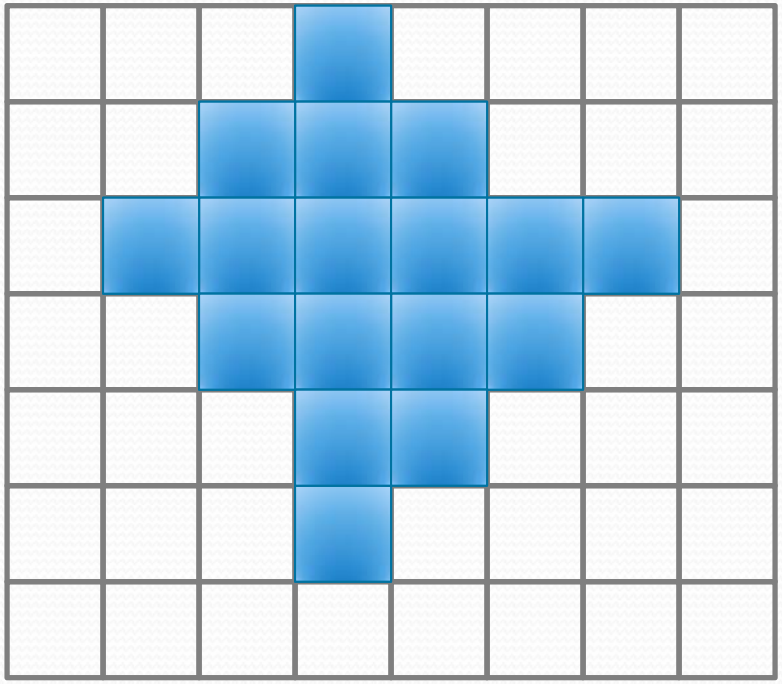
(aka)Rasterisation

- Given a polygon
  - E.g. vertices are in a clockwise fashion
- Pixelation : how to pixelize and draw it on the screen buffer



Scan Conversion Algorithm

- Given a polygon
  - E.g. it's vertices in a clockwise fashion
    - E.g (2,5) → (4,7) → (7,5) → (4,2)
  - Remember that vertices are given in this form!!!



Scan Convert Algorithm

- “Paint” every polygon

Importance of SCA

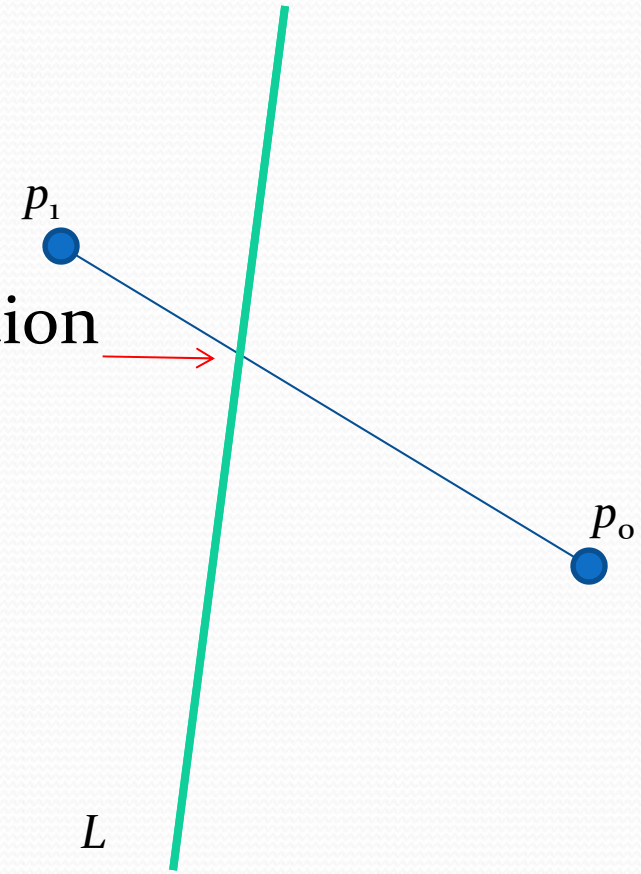
- Not only useful for painting a polygon
- A foundation for other algorithms in graphics
  - Hidden surface removal (L6)
  - Shading (L7)
  - Texture mapping (L11), etc.
- Terminology
  - Line segments = edges of a polygon

Intersection Computation

But first.. Let's review some math

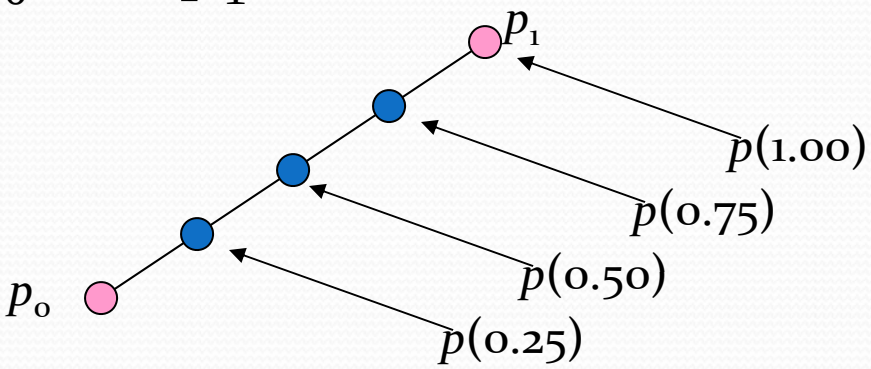
Line Intersection

- In general, given one line segment with two endpoints
  - Let $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$
- And an infinite line
  - $L : ax + by + c = 0$
- We want to compute their intersection



Straight Line Segments

- A straight line segment is the line joining two end points p_0 and p_1 :



- Any point on the line segment can be expressed in the form of

$$p(t) = (1\text{-}t)\, p_0 + t\, p_1$$

- This is called the ***linear interpolation*** of p_0 and p_1

The Parametric (Explicit) Form of a Line Segment

- Let $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$
- We can write the point as

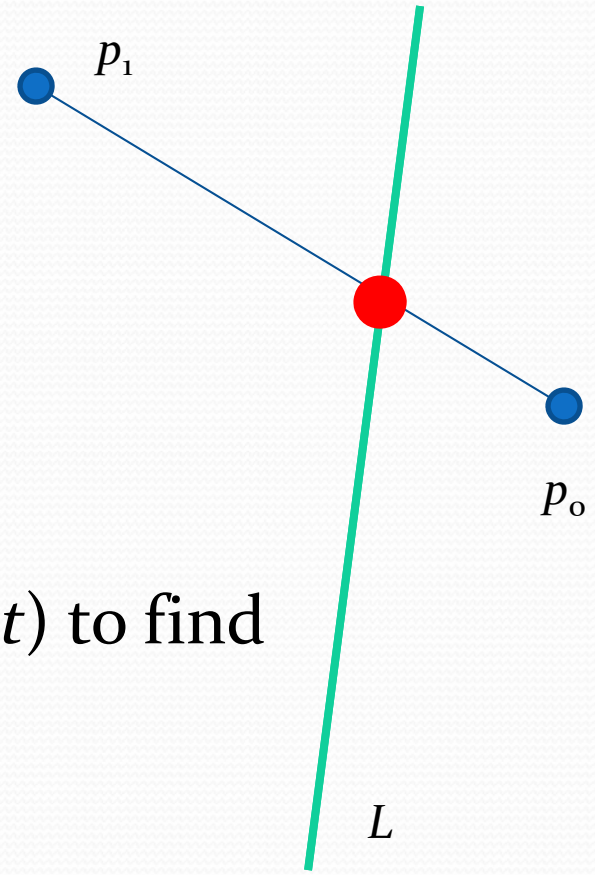
$$p(t) = (x(t), y(t))$$

  - for $x(t) = (1-t)x_0 + tx_1$ and $y(t) = (1-t)y_0 + ty_1$

- This is called the ***parametric form*** (or the ***explicit form***) of a curve if we can express the point with a ***parameter*** t
- Usually t is from 0 to 1

Line Intersection

- Given one infinite line and one line segment
  - Let $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$
  - $L : ax + by + c = 0$
- Substitute $p(t) = (x(t), y(t))$ into L
  - $a\,x(t) + b\,y(t) + c = 0$
  - $a\,[(1-t)x_0 + tx_1] + b[(1-t)y_0 + ty_1] + c = 0$
- A linear equation in t
- Solve t and substitute it back into $p(t)$ to find the intersection

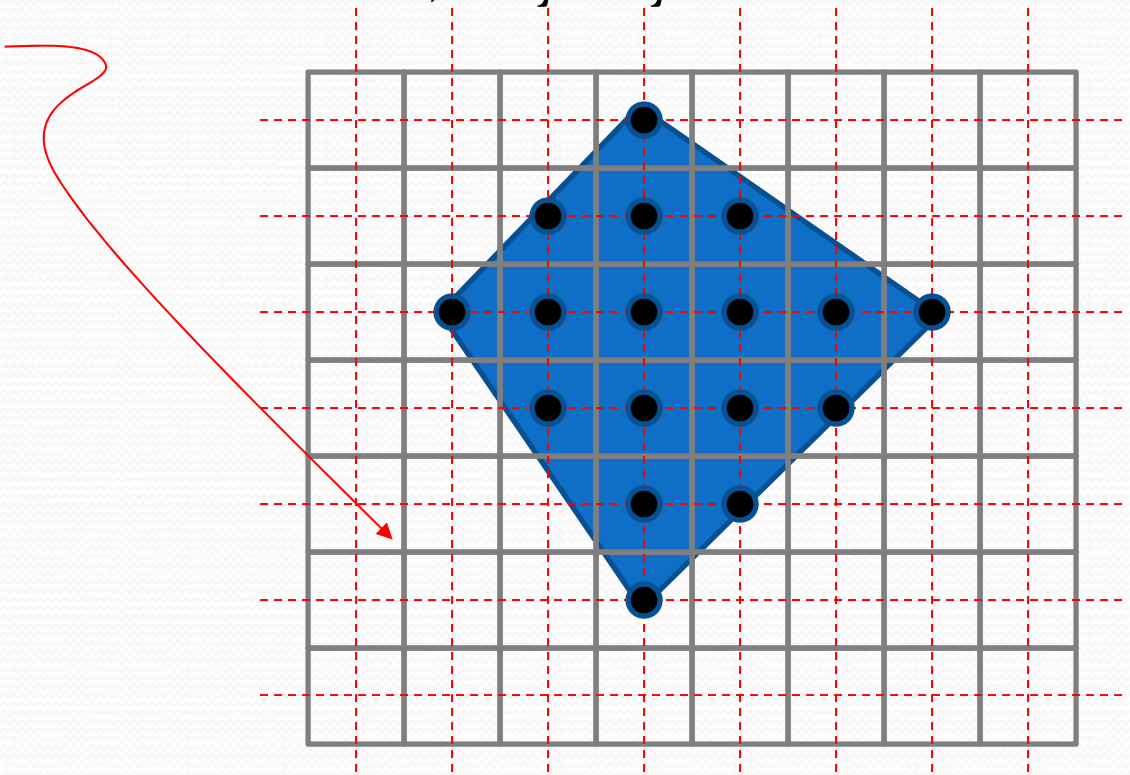


The SCA

(Rasterizstion, or pixelation)

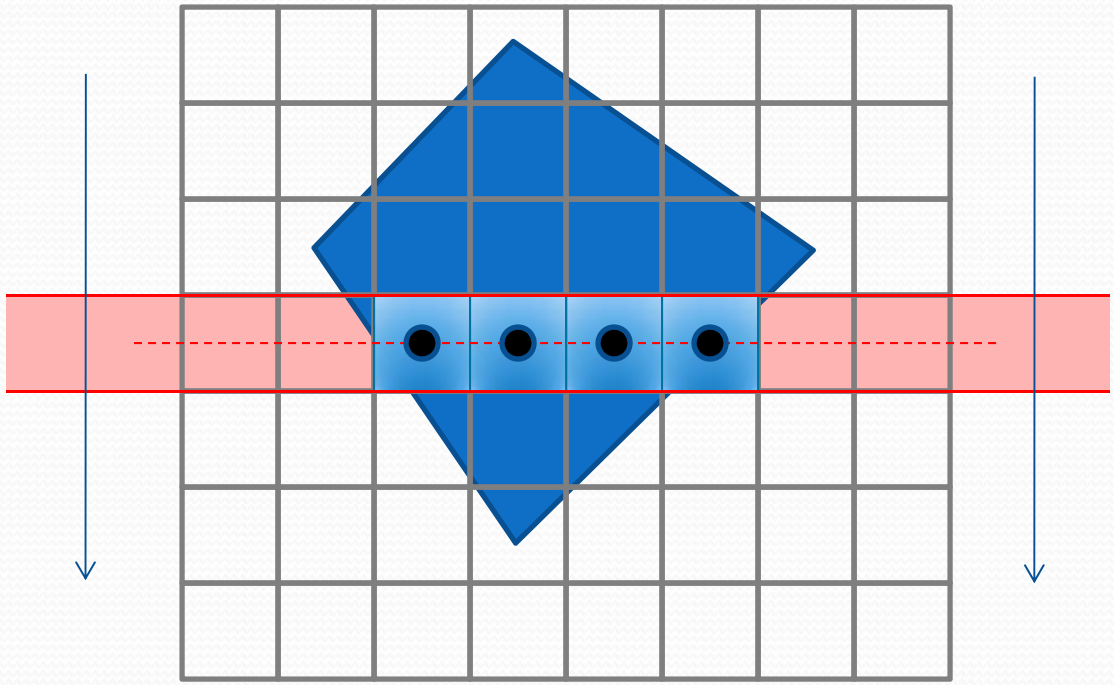
Scan Conversion Algorithm

- Why a certain pixel is turned on?
  - Criteria: the midpoint of that pixel is inside the polygon
    - (For some other textbooks, they may consider the corner instead)



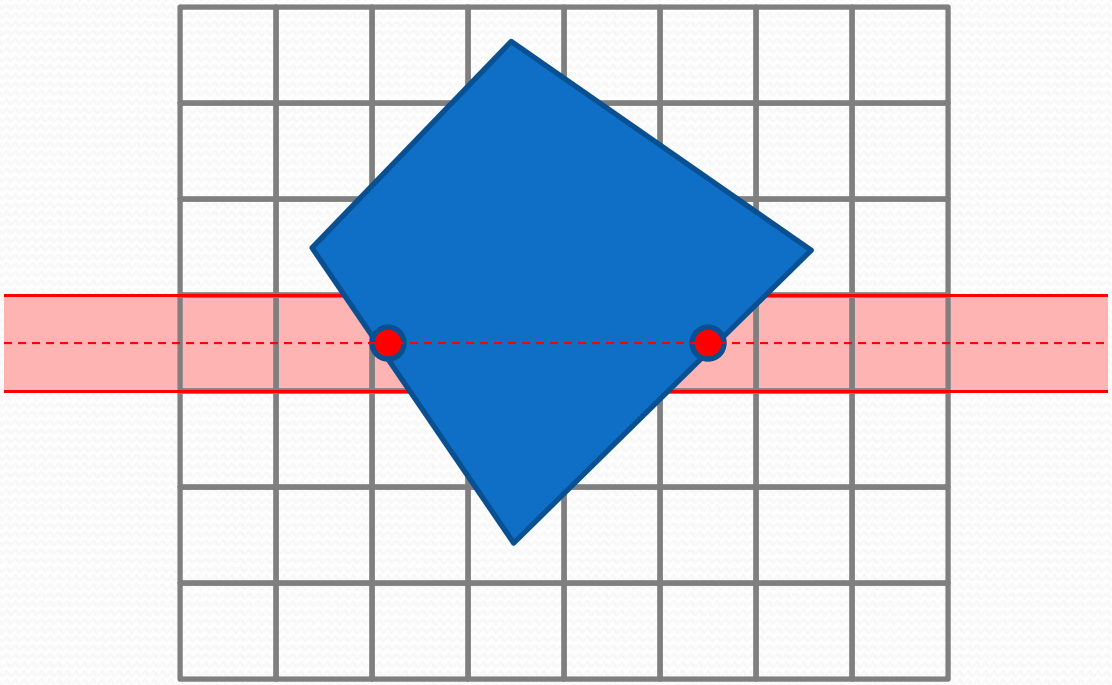
Scan Conversion Algorithm

- Each round, from top to bottom
  - Do one horizontal line, namely, one **scanline**
- Determine the leftmost and rightmost positions of each scanline
- Color the line



Scan Conversion Algorithm

- How to find the leftmost and rightmost positions of a scanline intersecting with the polygon edges?
  - Naïve way: for every scanline, try to compute the intersection of it with every edge of the polygon

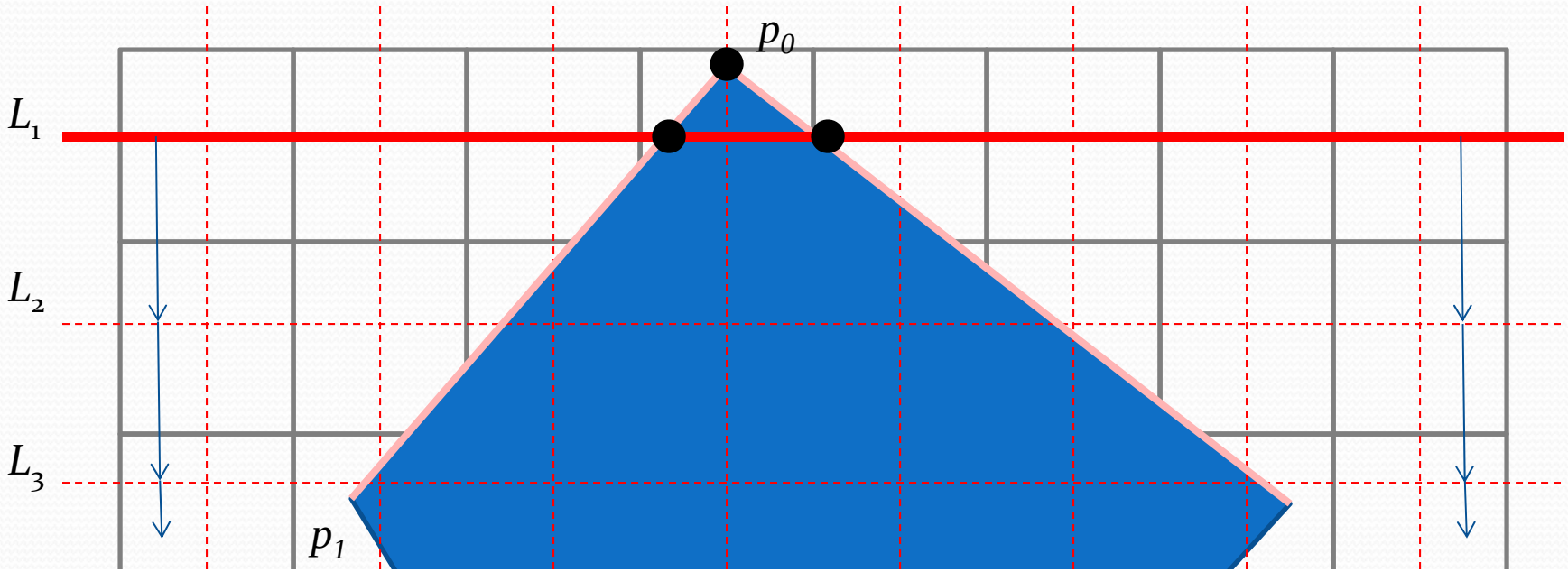


Scan Conversion Algorithm

- Remember that all vertices are given in a list of pairs of numbers
  - And if you have so many vertices…
  - NO!

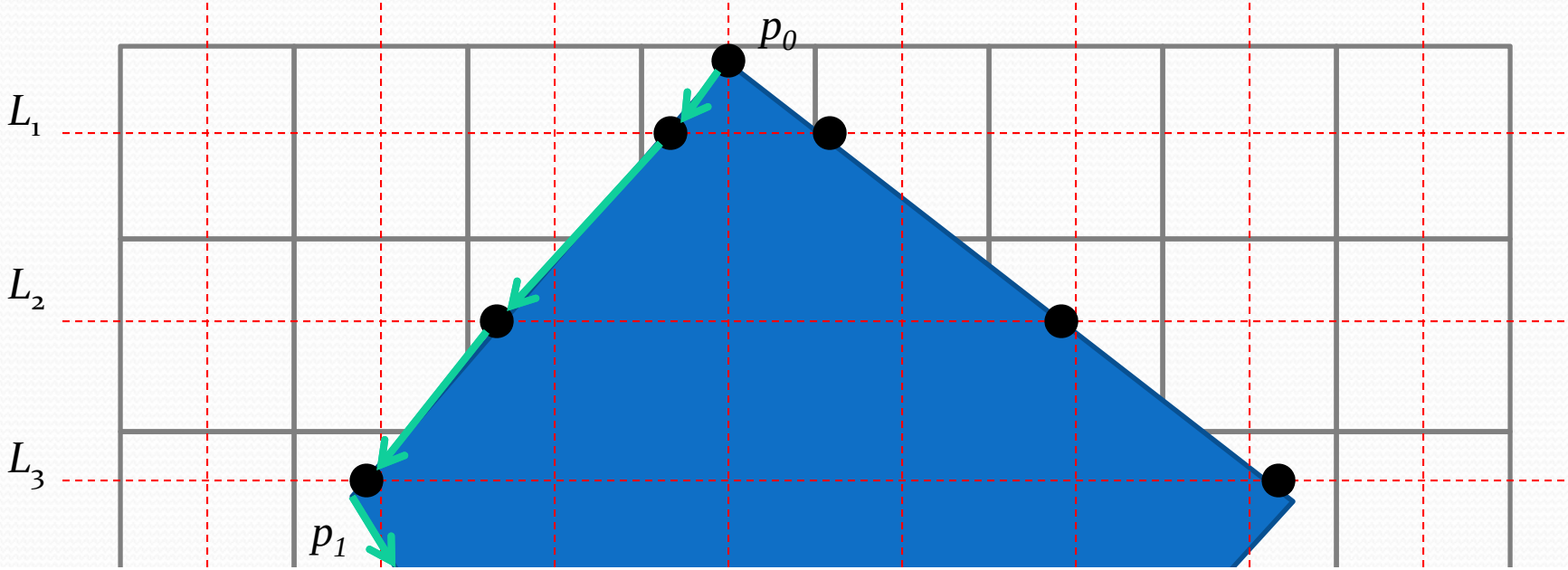
Scan Conversion Algorithm

- From the top most vertex p_0 of the polygon
  - You know which two segments are connected to p_0
  - The first scanline "must" be the one just below p_0
  - And it will (probably) intersect with the two edges that connects to p_0



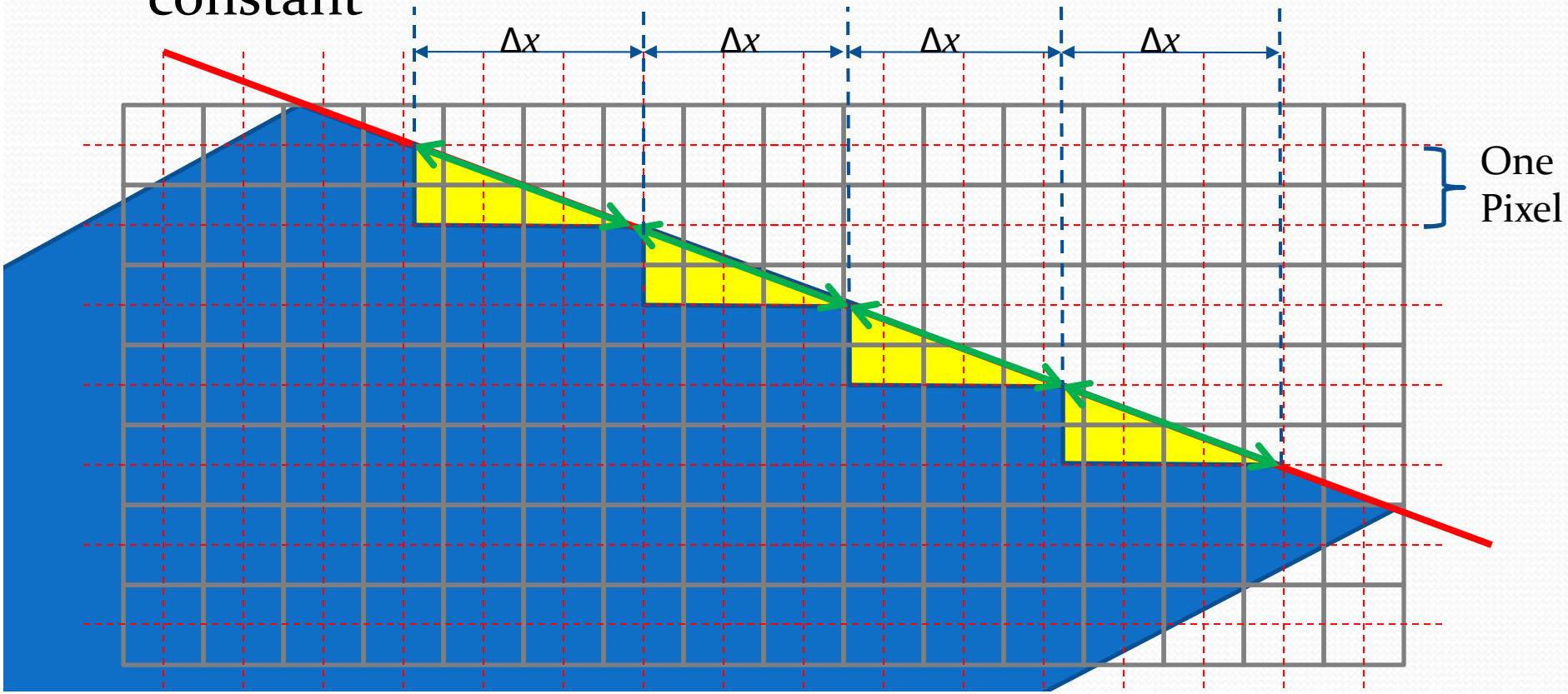
Scan Conversion Algorithm

- Let's assume that the scanline only intersects these two edges of the polygon
- Do we have to compute, i.e. calculate, the intersection of every scanline?
  - No! Because the amount of leftward movement is **constant**



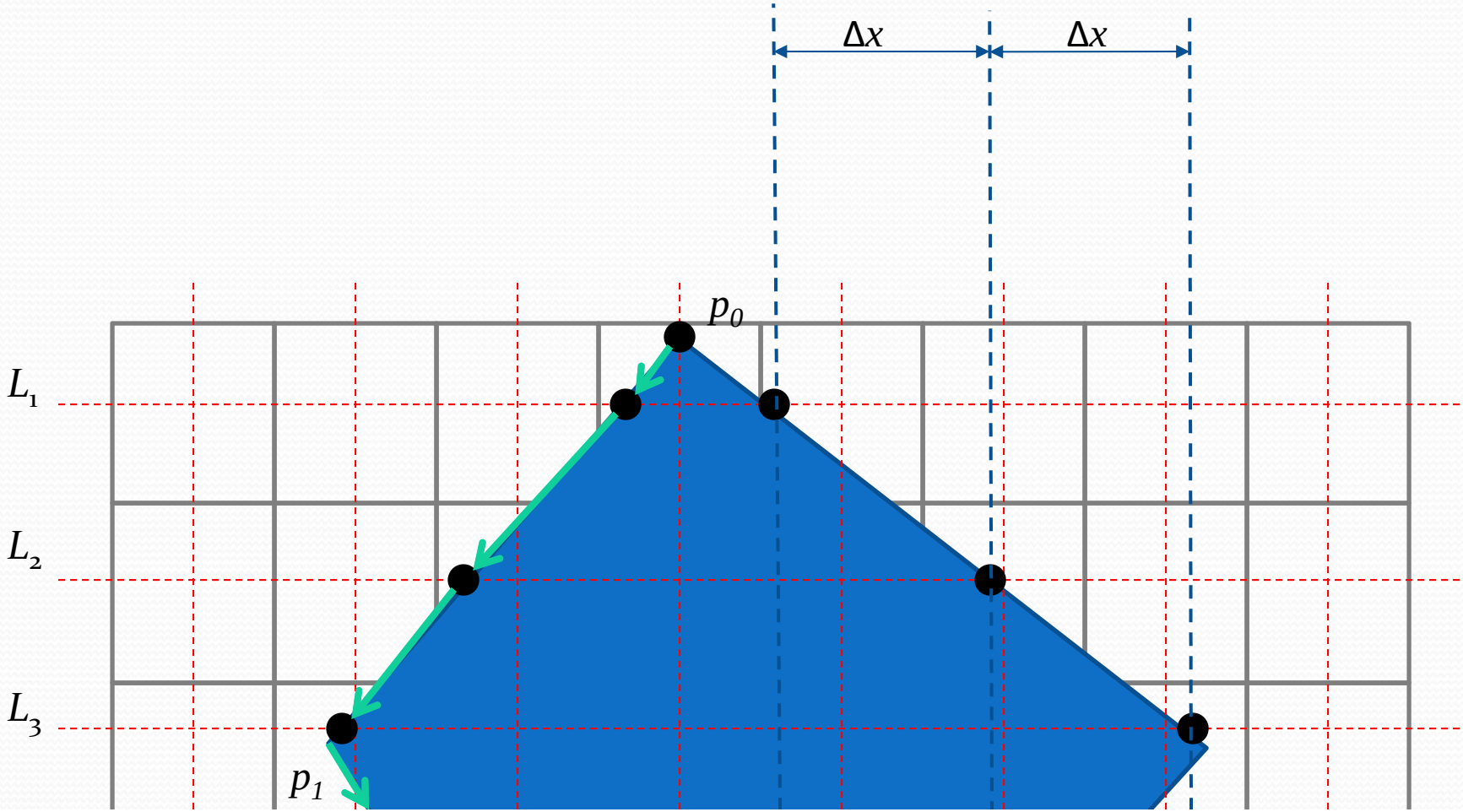
Constant Increment

- Each horizontal increment to the next scanline is **<u>constant(!!!)</u>**
- Because the slope and the downward increment is constant



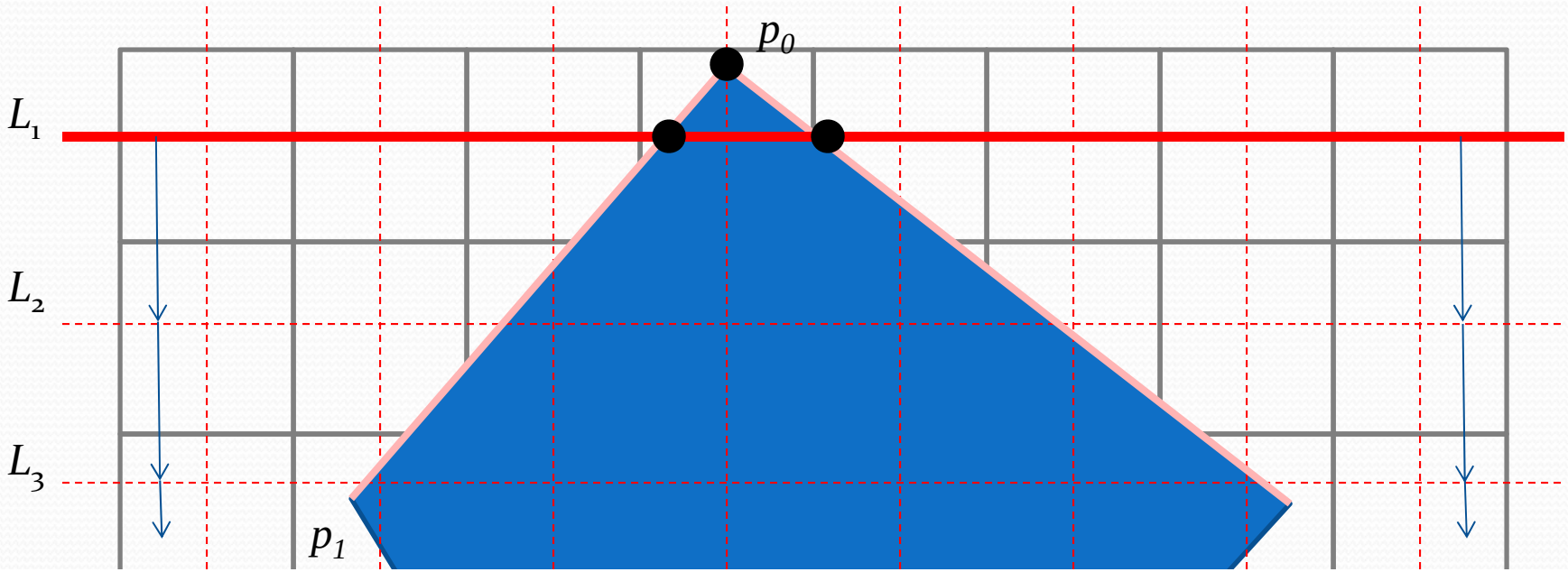
Scan Conversion Algorithm

- The amount of leftward movement is **constant**



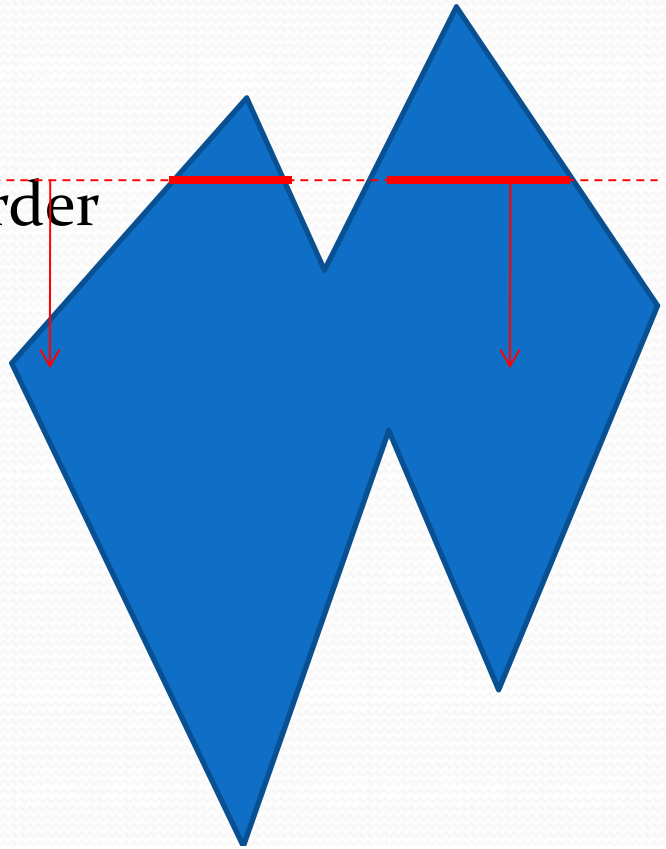
Scan Conversion Algorithm

- When the scanline is moving down, will it be always intersecting with those two edges?
  - If so, we can always know that which edges intersect with the scanline
  - NO? When?



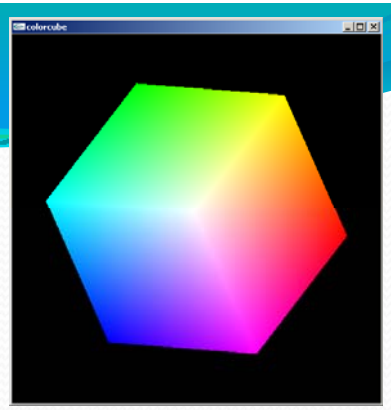
Consideration

- Could have more than one *section* of lines per scanline
  - Because of peaks and valleys
  - Solution: Maintain an **edge table**
- **Overall algorithm**
  - Sort all the vertices in the vertical order
  - Move the scanline from top
  - When the scanline reaches a vertex
    - A peak or valley
      - Manage the scanline segments
    - Otherwise
      - Switch to another polygon edge



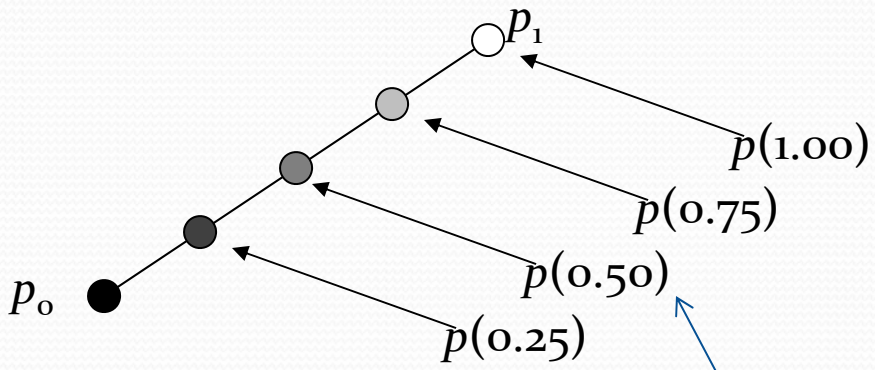
Interpolation of Colors

One of the Application of SCA



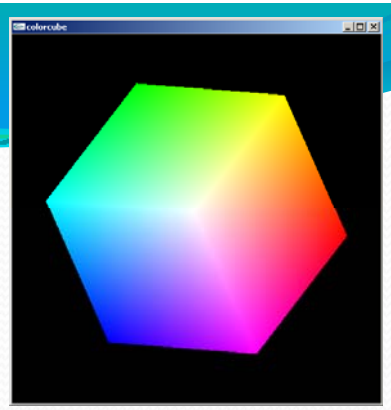
Interpolation of Colors

- Given a line segment $p(t) = (1\text{-}t)\, p_0 + t\, p_1$
  - With end points p_0, p_1

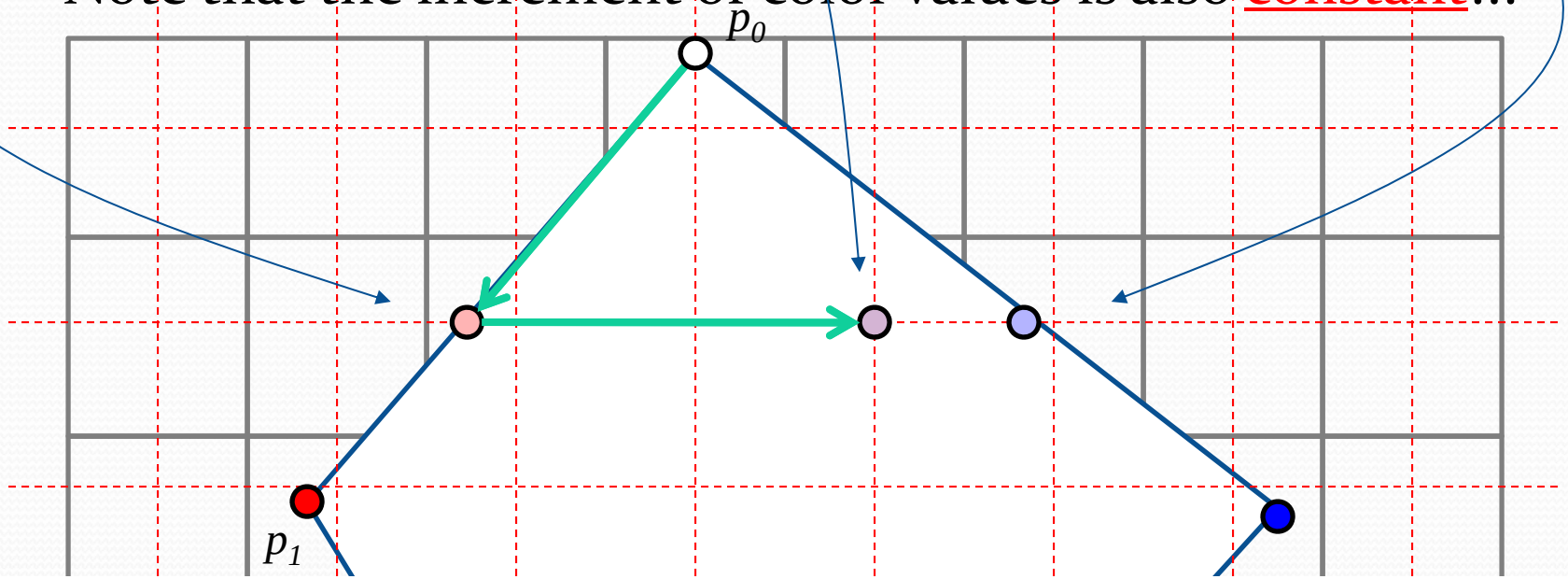


- If the colors of the two vertices of a line segment are different
  - $c_0 = (r_0, g_0, b_0)$ at p_0 and $c_1 = (r_1, g_1, b_1)$ at p_1
- The color of any point at t is $c(t) = (1\text{-}t)\, c_0 + t\, c_1$

Interpolation of Colors



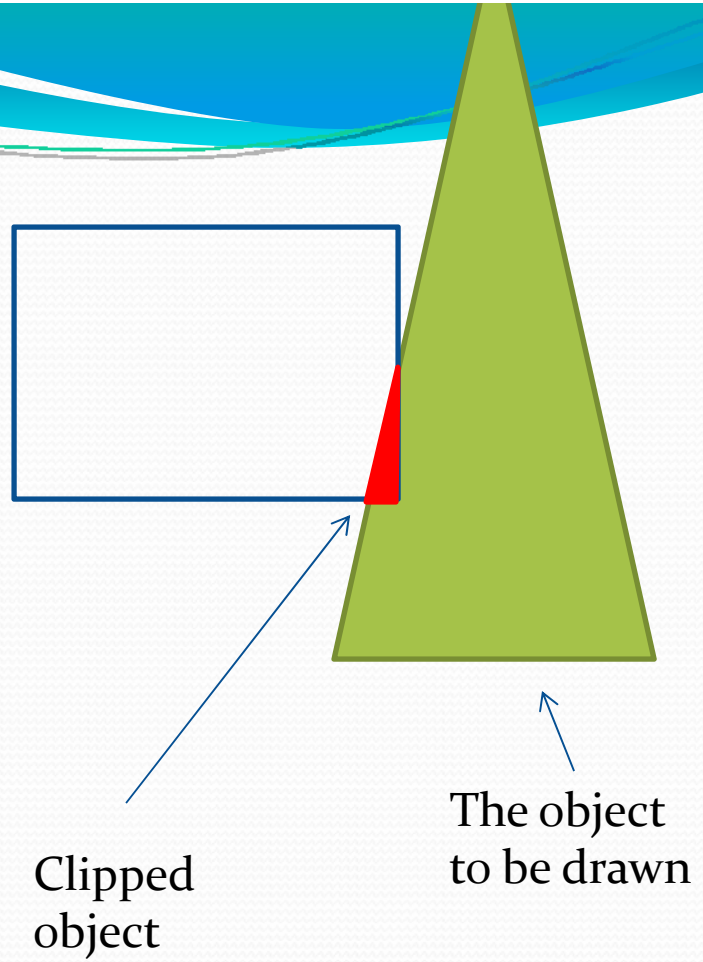
- For a point on an edge, interpolate its color by the two end points of that edge
  - Afterwards, interpolate the colors of the points between two points on the two edges
  - Note that the increment of color values is also constant!!!



Clipping

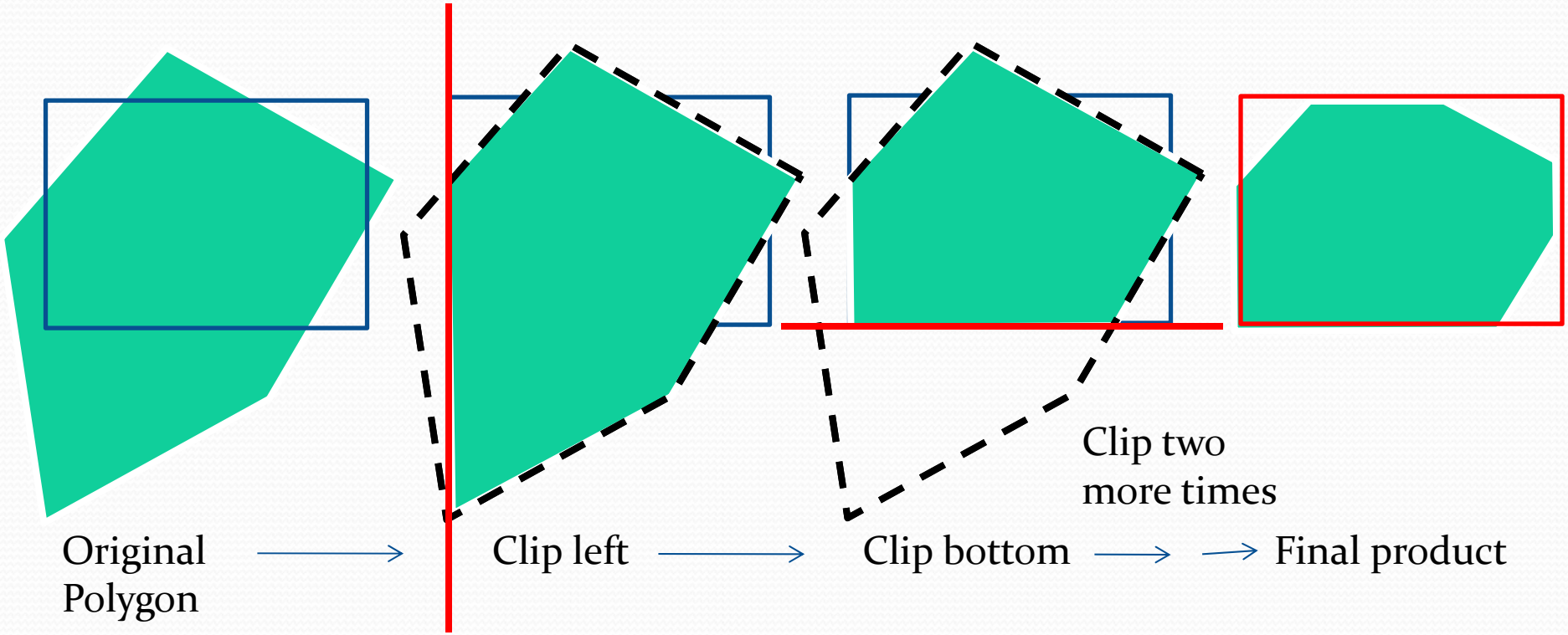
Clipping

- What if an object is only partially seen inside the visible screen area?
- If we scan-convert the whole object, it will be a big waste of time to go through all the out-of-screen pixels
- Solution
  - Do **clipping** before scan conversion



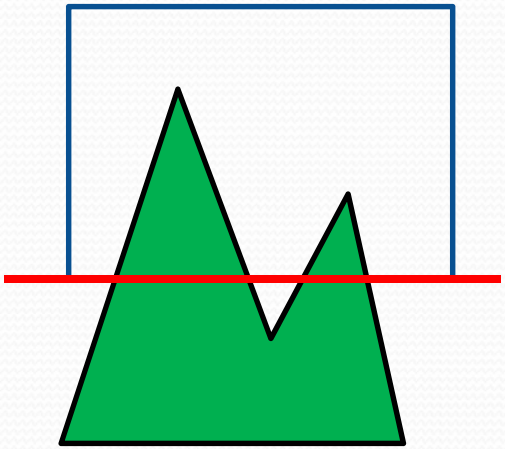
Clipping

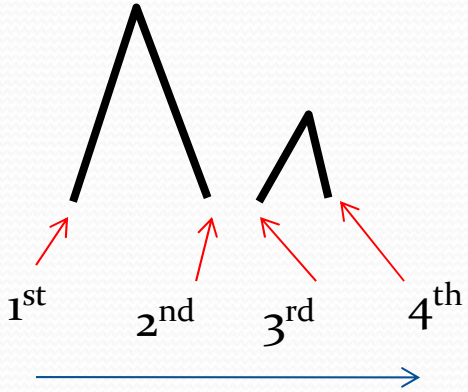
- Clip the polygon with one side of the screen at a time
  - 4 times, left, right, top and bottom

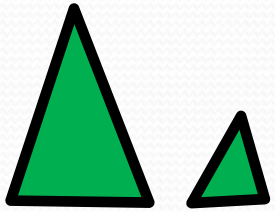


More Complicated Cases

- More than two intersections with one single clipping line
- Clip all the line segments and leave them "open" with the clipping line
- Join every i^{th} and $(i+1)^{th}$ intersections for i is odd
- (Nasty cases: when the polygon has horizontal or vertical edges that lie exactly on the edges of the screen)







Purposes of Clipping and SCA

- Two essential topics in the graphic pipeline
- However, they are usually implemented and hidden in most graphics library/hardware
- And, however again, we need to know where do they exist
- More importantly, SCA is important for some later topics, e.g. shading and texture mapping

Announcement

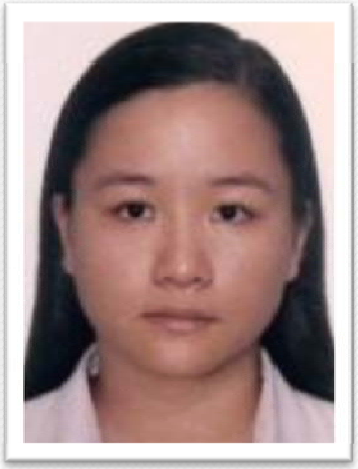
Some Admin

- Tutorials and Labs already started

Phi Long
(Wed)

Alan
(Fri)

LEONG KAI KHEE



TRUONG CHAU THY
(Kathy)

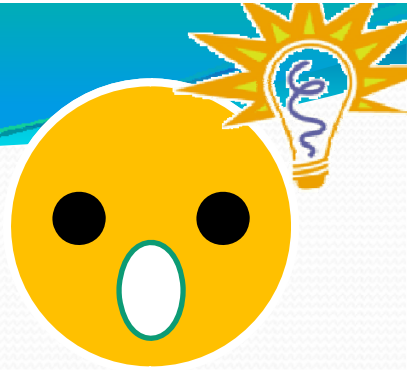
FENDY GOUW

ZHAO YANG

Labs and Homework

- Settle lab and tutorial slots
  - Please confirm your lab slots
  - Email and tell your tutor about your tutorial changes
  - Prepare answers for participation
- **Assignment 1** is out
  - Warm up with OpenGL
    - But please don't overstretch yourself. Do it at your own pace.

About Compiling OpenGL



- I tried other compilers, g++, Bloodshed Dev-C++
  - They may work but they are very complicated
- Please use Microsoft Visual Studio 8.0 or 10.0
  - You can download it from ELMS:
    - http://msdn70.e-academy.com/socnusingapore_cs
    - …if you are taking SOC modules
  - If you failed to download that, please go to or send email to technical service
    - helpdesk@comp.nus.edu
    - CC me to confirm your request

Marking Scheme of Lab Assignments

- 20% for 5 lab assignments, 4% for each
- Marking scheme for EACH assignment
  - 20% for compilation and programming style
    - Just neat, (NOT a high requirement), + a bit comments
    - Basically just mean that it can be compiled using the **lab machines**
  - 40% for requirement stated in the assignment
  - 20% for extra features NOT taught in lab/class
    - Number of features x 10%
  - 20% for artistic appreciation
    - 0% It doesn't make any sense
    - 5% Ok
    - 10% Good looking (Most (70%) of students will be in this)
    - 15% Very good, exceptional
    - 20% The best 5 students

The Top 20 replies by programmers when their programs do not work:

20. "That's weird..."
19. "It's never done that before."
18. "It worked yesterday."
17. "How is that possible?"
16. "It must be a hardware problem."
15. "What did you type in wrong to get it to crash?"
14. "There is something funky in your data."
13. "I haven't touched that module in weeks!"
12. "You must have the wrong version."
11. "It's just some unlucky coincidence."
10. "I can't test everything!"
9. "THIS can't be the source of THAT."
8. "It works, but it hasn't been tested."
7. "Somebody must have changed my code."
6. "Did you check for a virus on your system?"
5. "Even though it doesn't work, how does it feel?
4. "You can't use that version on your system."
3. "Why do you want to do it that way?"
2. "Where were you when the program blew up?"

No excuse for submitting HW wrongly

And the Number One reply by programmers when their programs don't work:

Facebook

- Remember to visit the CS3241 Facebook page
  - Search for NUSCG