CS380: Computer Graphics
Screen Space & World Space

Sung-Eui Yoon
(윤성의)

Course URL:
http://sglab.kaist.ac.kr/~sungeui/CG

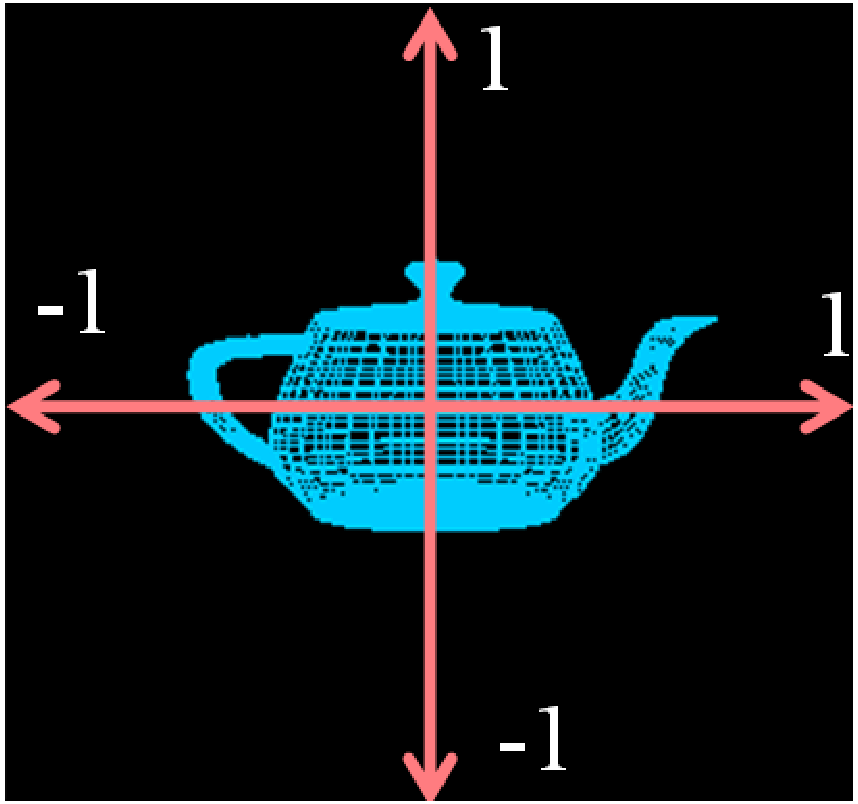
KAIST

Class Objectives

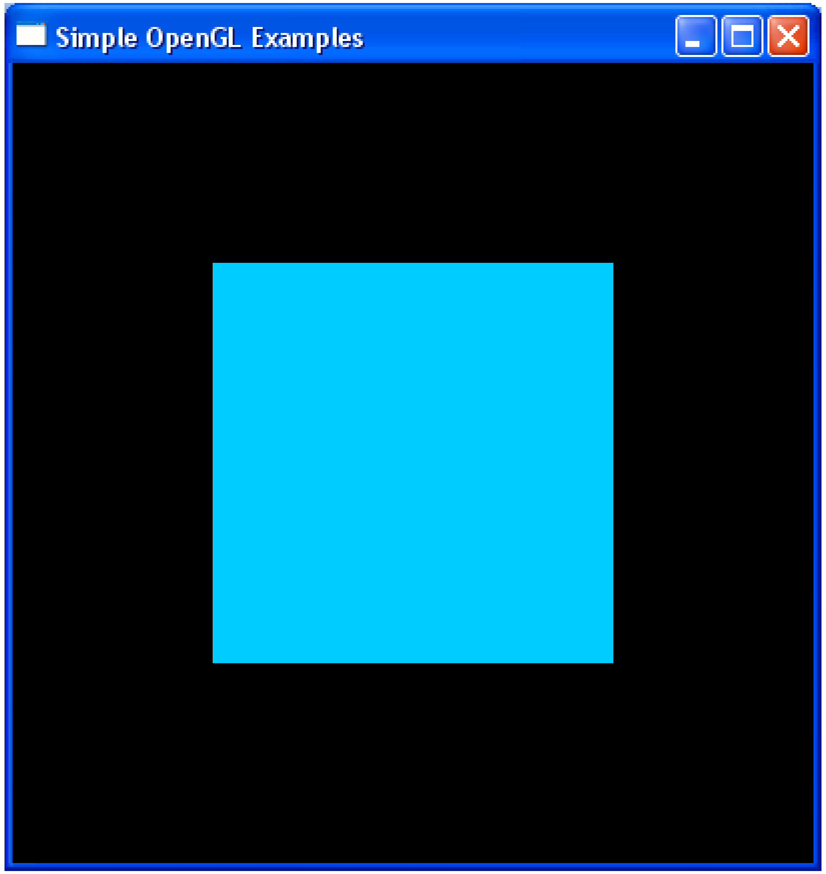
- Understand different spaces and basic OpenGL commands
- Understand a continuous world, Julia sets

Your New World

- A 2D square ranging from (-1, -1) to (1, 1)
- You can draw in the box with just a few lines of code



Code Example



OpenGL Code:

```
glColor3d(0.0, 0.8, 1.0);

glBegin(GL_POLYGON);
      glVertex2d(-0.5, -0.5);
      glVertex2d( 0.5, -0.5);
      glVertex2d( 0.5,  0.5);
      glVertex2d(-0.5,  0.5);
glEnd();
```

OpenGL Command Syntax

- **glColor3d(0.0, 0.8, 1.0);**

Suffix	Data Type	Corresponding C-Type	OpenGL Type
b	8-bit int.	singed char	GLbyte
s	16-bit int.	short	GLshort
i	32-bit int.	int	GLint
f	32-bit float	float	GLfloat
d	64-bit double	double	GLdouble
ub	8-bit unsinged int.	unsigned char	GLubyte
us	16-bit unsigned int.	unsigned short	GLushort
ui	32-bit unsigned int.	unsigned int	GLuint

OpenGL Command Syntax

- **You can use pointers or buffers**

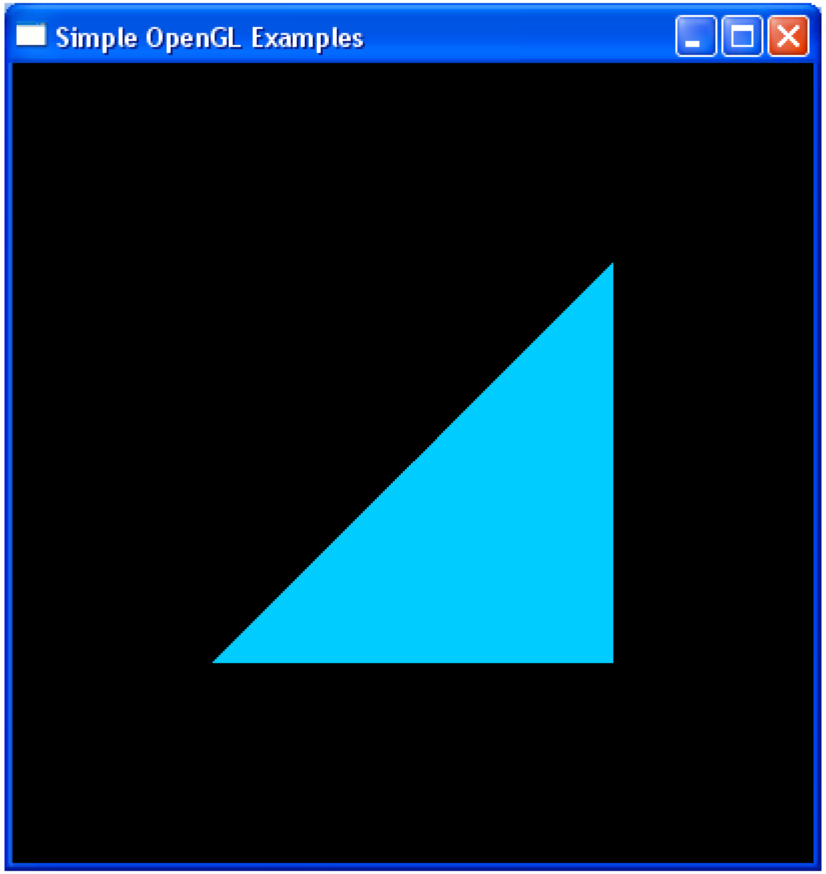
```
glColor3f(0.0, 0.8, 1.0);

GLfloat color_array [] = {0.0, 0.8, 1.0};
glColor3fv (color_array);
```

- **Using buffers for drawing is much more efficient**

KAIST

Another Code Example



OpenGL Code:

```
glColor3d(0.0, 0.8, 1.0);

glBegin(GL_POLYGON);
        glVertex2d(-0.5, -0.5);
        glVertex2d( 0.5, -0.5);
        glVertex2d( 0.5,  0.5);
glEnd()
```

Drawing Primitives in OpenGL

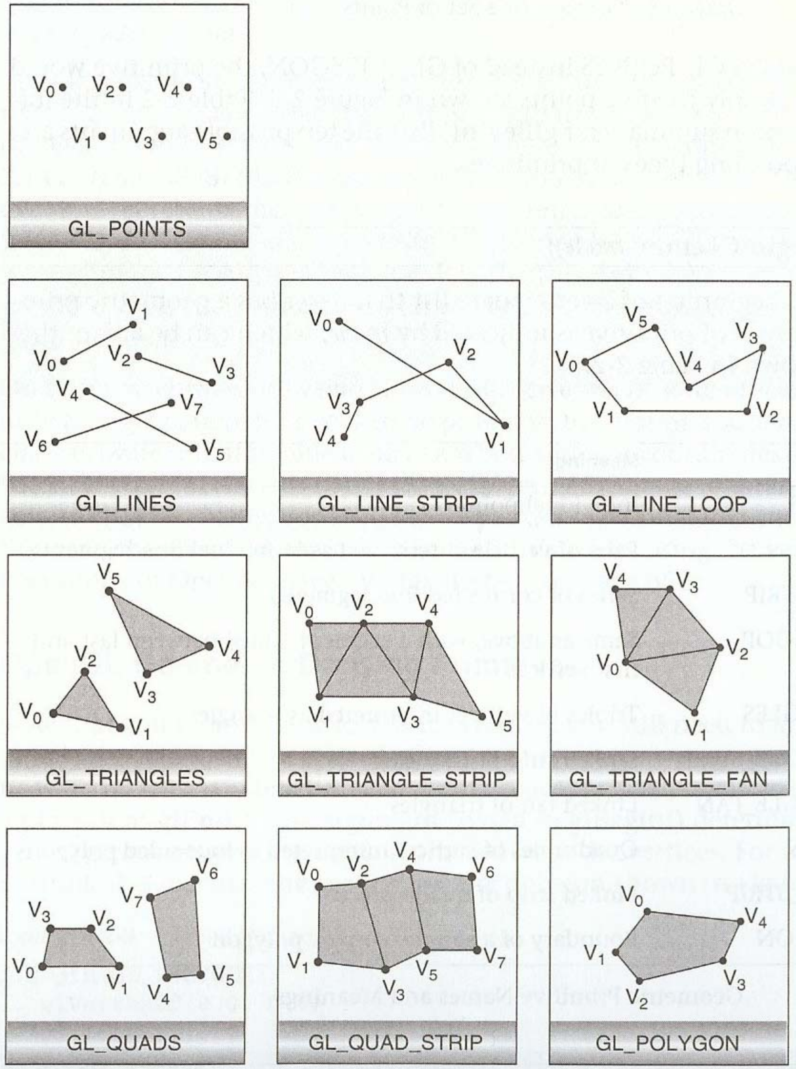
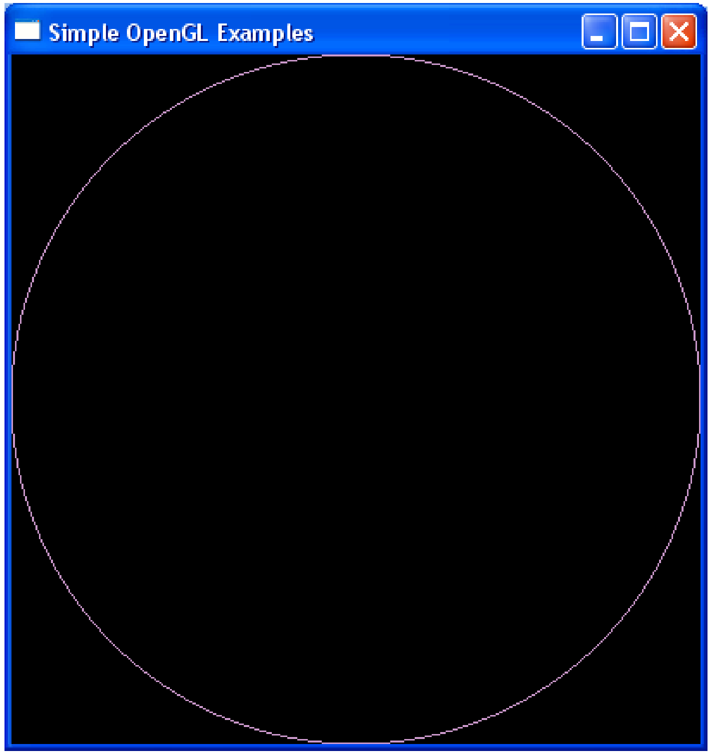


Figure 2-7 Geometric Primitive Types

The red book

Yet Another Code Example



OpenGL Code:

```
glColor3d(0.8, 0.6, 0.8);


glBegin(GL_LINE_LOOP);
for (i = 0; i < 360;i = i + 2)
{
   x = cos(i*pi/180);
   y = sin(i*pi/180);
   glVertex2d(x, y);
}
glEnd();
```

OpenGL as a State Machine

- **OpenGL maintains various states until you change them**

```
// set the current color state
glColor3d(0.0, 0.8, 1.0);

glBegin(GL_POLYGON);
     glVertex2d(-0.5, -0.5);
     glVertex2d( 0.5, -0.5);
     glVertex2d( 0.5,  0.5);
glEnd()
```

OpenGL as a State Machine

- OpenGL maintains various states until you change them

- Many state variables refer to modes (e.g., lighting mode)
  - You can enable, glEnable (), or disable, glDisable ()

- You can query state variables
  - glGetFloatv (), glIsEnabled (), etc.
  - glGetError (): very useful for debugging

KAIST

Debugging Tip

```
#define CheckError(s)                                   \
 {                                                      \
   GLenum error = glGetError();                         \
   if (error)                                           \
     printf("%s in %s\n",gluErrorString(error),s);      \
 }
```

```
glTexCoordPointer (2, x,  sizeof(y), (GLvoid *) TexDelta);
CheckError ("Tex Bind");

glDrawElements(GL_TRIANGLES, x, GL_UNSIGNED_SHORT, 0);
CheckError ("Tex Draw");
```

OpenGL Ver. 4.3 (Using Retained Mode)

```
#include <iostream>
using namespace std;
#include "vgl.h"
#include "LoadShaders.h"
enum VAO_IDs { Triangles, NumVAOs };
enum Buffer_IDs { ArrayBuffer, NumBuffers };
enum Attrib_IDs { vPosition = 0 };
GLuint VAOs[NumVAOs];
GLuint Buffers[NumBuffers];
const GLuint NumVertices = 6;

Void init(void)  {
glGenVertexArrays(NumVAOs, VAOs);
glBindVertexArray(VAOs[Triangles]);
GLfloat vertices[NumVertices][2] = {
{ -0.90, -0.90 }, // Triangle 1
{ 0.85, -0.90 },
{ -0.90, 0.85 },
{ 0.90, -0.85 }, // Triangle 2
{ 0.90, 0.90 },
{ -0.85, 0.90 } };
glGenBuffers(NumBuffers, Buffers);

glBindBuffer(GL_ARRAY_BUFFER, Buffers[ArrayBuffer]);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices),
vertices, GL_STATIC_DRAW);
ShaderInfo shaders[] = {
{ GL_VERTEX_SHADER, "triangles.vert" },
{ GL_FRAGMENT_SHADER, "triangles.frag" },
{ GL_NONE, NULL } };
GLuint program = LoadShaders(shaders);
glUseProgram(program);
glVertexAttribPointer(vPosition, 2, GL_FLOAT,
GL_FALSE, 0, BUFFER_OFFSET(0));
glEnableVertexAttribArray(vPosition);
}

Void display(void) {
glClear(GL_COLOR_BUFFER_BIT);
glBindVertexArray(VAOs[Triangles]);
glDrawArrays(GL_TRIANGLES, 0, NumVertices);
glFlush();
}
Int main(int argc, char** argv) {
glutInit(&argc, argv);   glutInitDisplayMode(GLUT_RGBA);
glutInitWindowSize(512, 512);
glutInitContextVersion(4, 3);
glutInitContextProfile(GLUT_CORE_PROFILE);
glutCreateWindow(argv[0]);
if (glewInit()) {
exit(EXIT_FAILURE); }
init();glutDisplayFunc(display);   glutMainLoop();
}
```

Julia Sets (Fractal)

Demo

- Study a visualization of a simple iterative function defined over the imaginary plane
- It has chaotic behavior
  - Small changes have dramatic effects

Julia Set - Definition

- **The Julia set J_c for a number c in the complex plane P is given by:**

$$J_c = \{ p \mid p \in P \text{ and } p_{i+1} = p_i^2 + c \text{ converges to a fixed limit } \}$$

KAIST

Complex Numbers

- **Consists of 2 tuples (Real, Imaginary)**
  - **E.g., $c = a + bi$**

- **Various operations**
  - $c_1 + c_2 = (a_1 + a_2) + (b_1 + b_2)i$
  - $c_1 \cdot c_2 = (a_1a_2 - b_1b_2) + (a_1b_2 + a_2b_1)i$
  - $(c_1)^2 = ((a_1)^2 - (b_1)^2) + (2\, a_1b_1)i$
  - $|c| = sqrt(a^2 + b^2)$

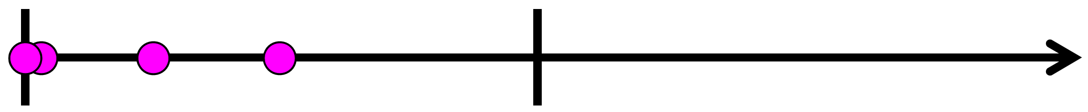
KAIST

Convergence Example

- **Real numbers are a subset of complex numbers:**
  - **Consider c = [0, 0], and p = [x, 0]**
  - **For what values of x is $x_{i+1} = x_i^2$ convergent?**

How about $x_0 = 0.5$?

x_{0-4} = 0.5, 0.25, **0.0625, 0.0039**

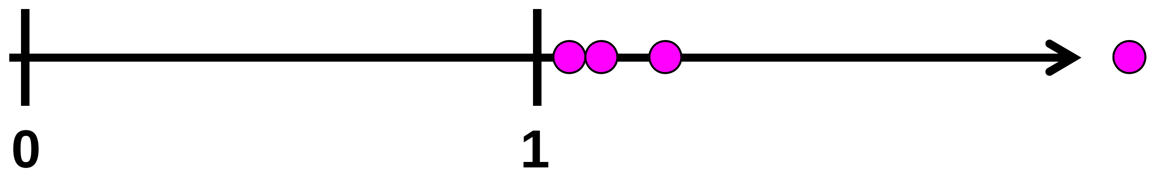


Convergence Example

- **Real numbers are a subset of complex numbers:**
  - **consider c = [0, 0], and p = [x, 0]**
  - **for what values of x is $x_{i+1} = x_i^2$ convergent?**

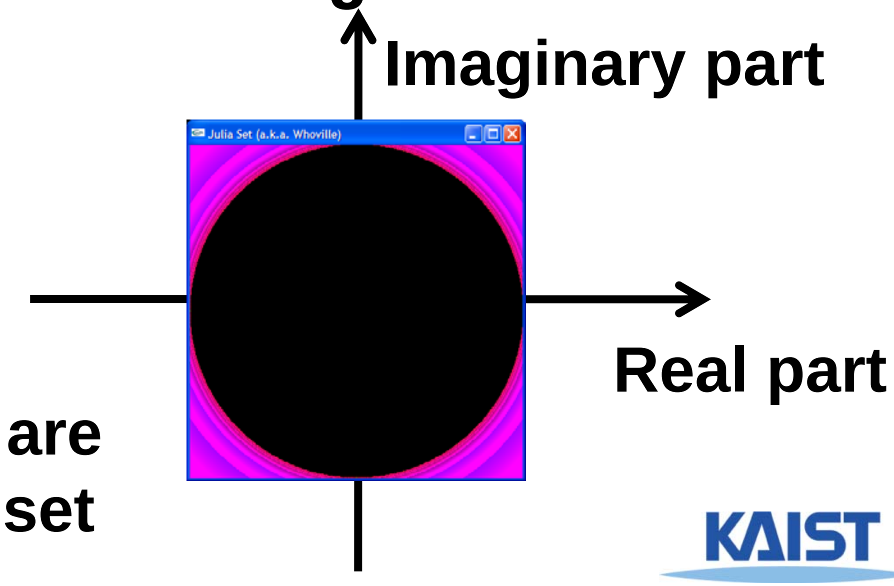
How about $x_0 = 1.1$?

x_{0-4} = 1.1, 1.21, 1.4641, 2.14358



Convergence Properties

- Suppose c = [0,0], for what complex values of p does the series converge?
- For real numbers:
  - If $|x_i| > 1$, then the series diverges
- For complex numbers
  - If $|p_i| > 2$, then the series diverges
  - Loose bound



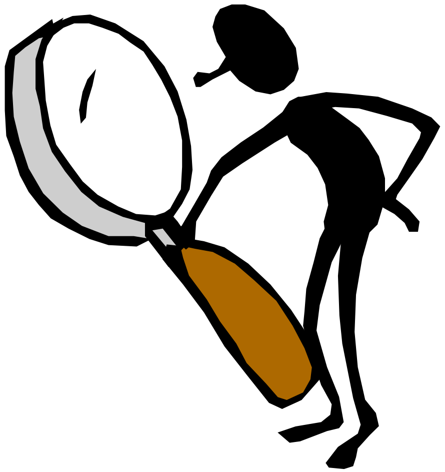
The black points are the ones in Julia set

A Peek at the Fractal Code

```
class Complex {
        float re, im;
};

viod Julia (Complex p, Complex c, int & i, float & r)
{
   int maxIterations = 256;
   for (i = 0; i < maxIterations;i++)
   {
        p = p*p + c;
        rSqr = p.re*p.re + p.im*p.im;

        if( rSqr > 4 )
            break;
    }
    r = sqrt(rSqr);
}
```



i & r are used to assign a color

How can we see more?

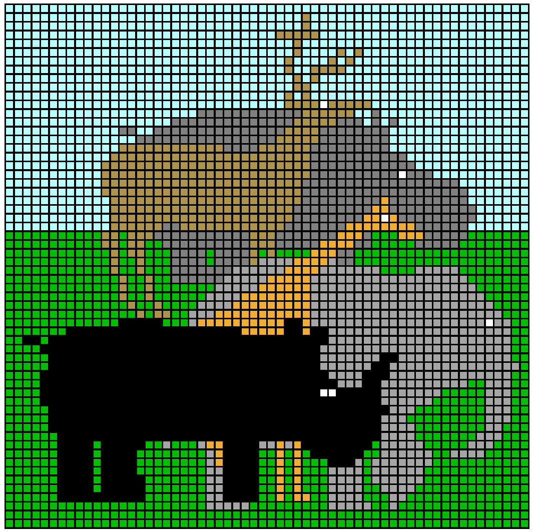
- Our world view allows us to see so much
  - What if we want to zoom in?
- We need to define a mapping from our desired world view to our screen

Mapping from World to Screen

Screen Space

- Graphical image is presented by setting colors for a set of discrete samples called “pixels”
  - Pixels displayed on screen in windows
- Pixels are addressed as 2D arrays
  - Indices are “screen-space” coordinates

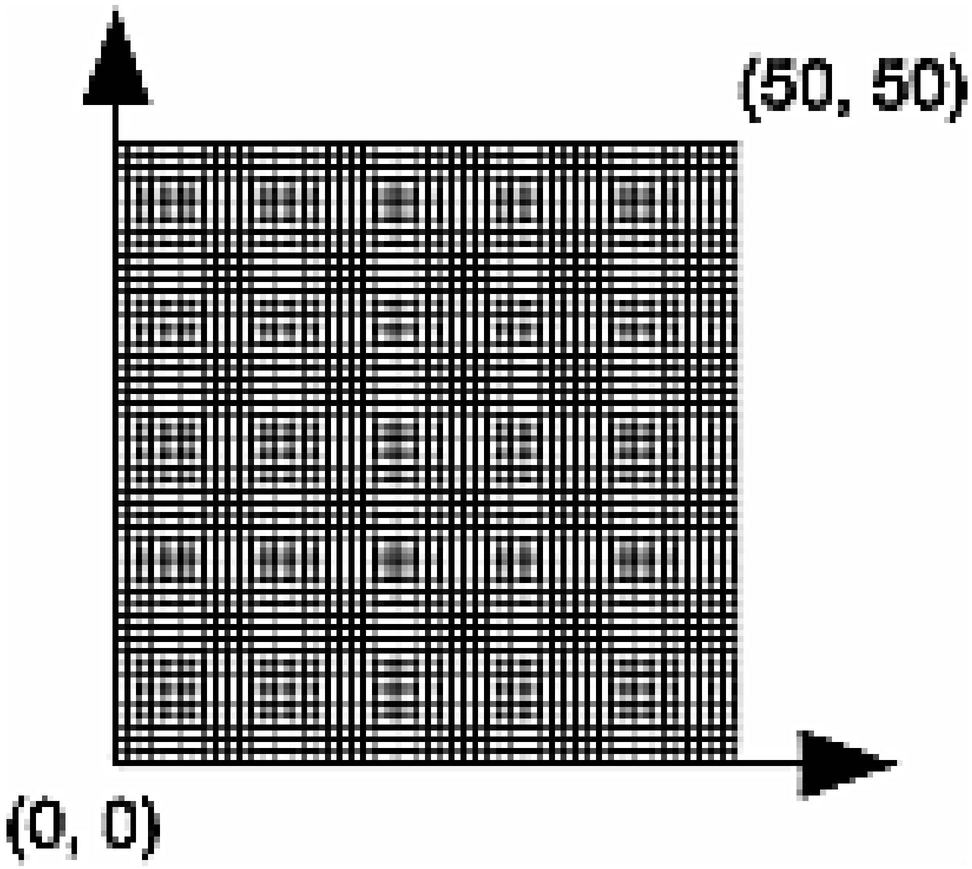
(0,0) (width-1,0)



(0,height-1) (width-1, height-1)

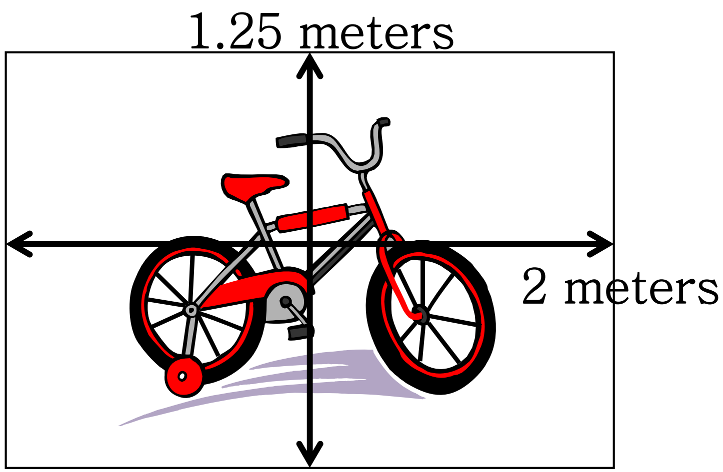
KAIST

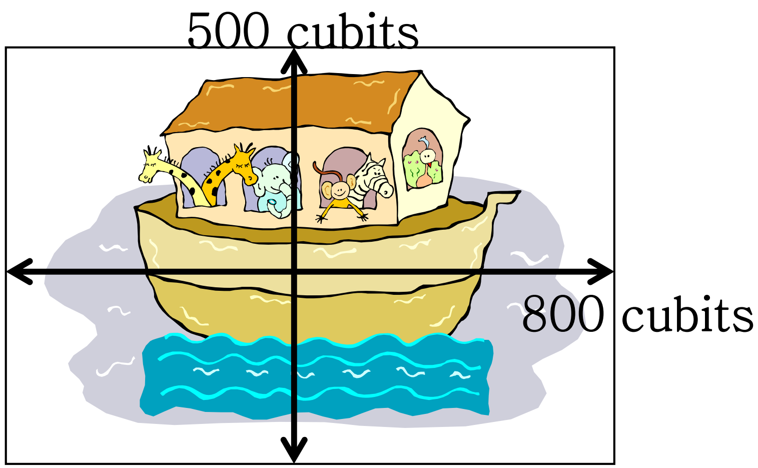
OpenGL Coordinate System



Pixel Independence

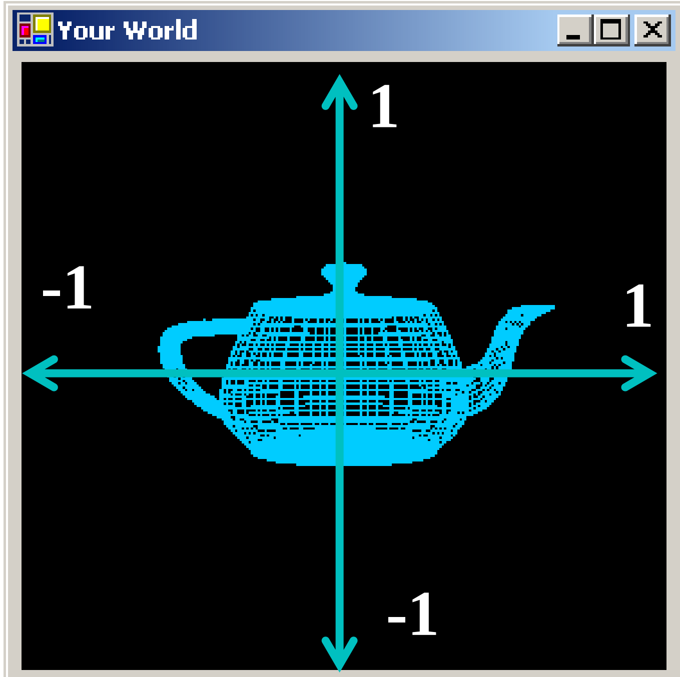
- Often easier to structure graphical objects independent of screen or window sizes
- Define graphical objects in "world-space"





Normalized Device Coordinates

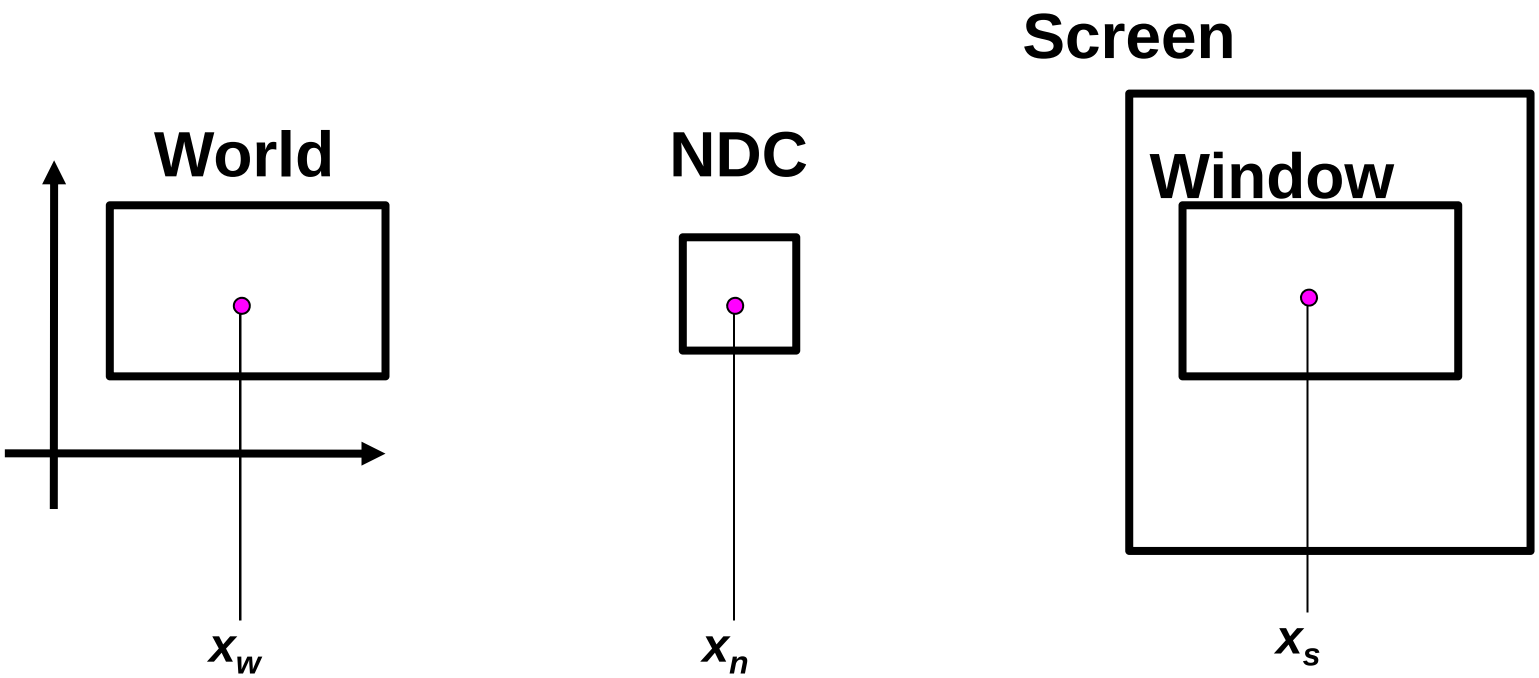
- Intermediate "rendering-space"
  - Compose world and screen space
- Sometimes called "canonical screen space"



Why Introduce NDC?

- Simplifies many rendering operations
  - Clipping, computing coefficients for interpolation
  - Separates the bulk of geometric processing from the specifics of rasterization (sampling)
  - Will be discussed later

Mapping from World to Screen



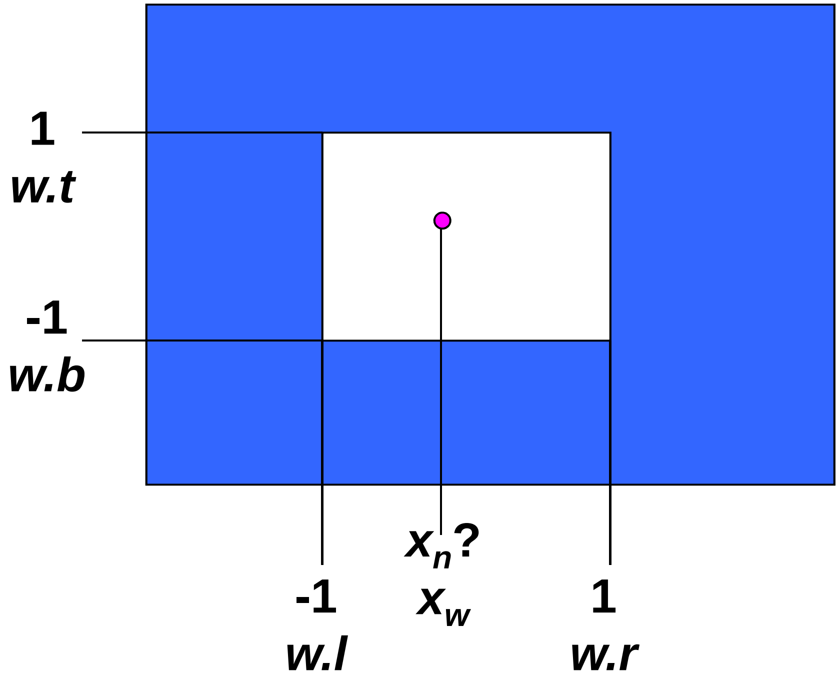
World Space to NDC

$$\frac{x_n - (-1)}{1 - (-1)} = \frac{x_w - (w.l)}{w.r - w.l}$$

$$x_n = 2\frac{x_w - (w.l)}{w.r - w.l} - 1$$

$$x_n = Ax_w + B$$

$$A = \frac{2}{w.r - w.l}, \quad B = -\frac{w.r + w.l}{w.r - w.l}$$



NDC to Screen Space

- **Same approach**

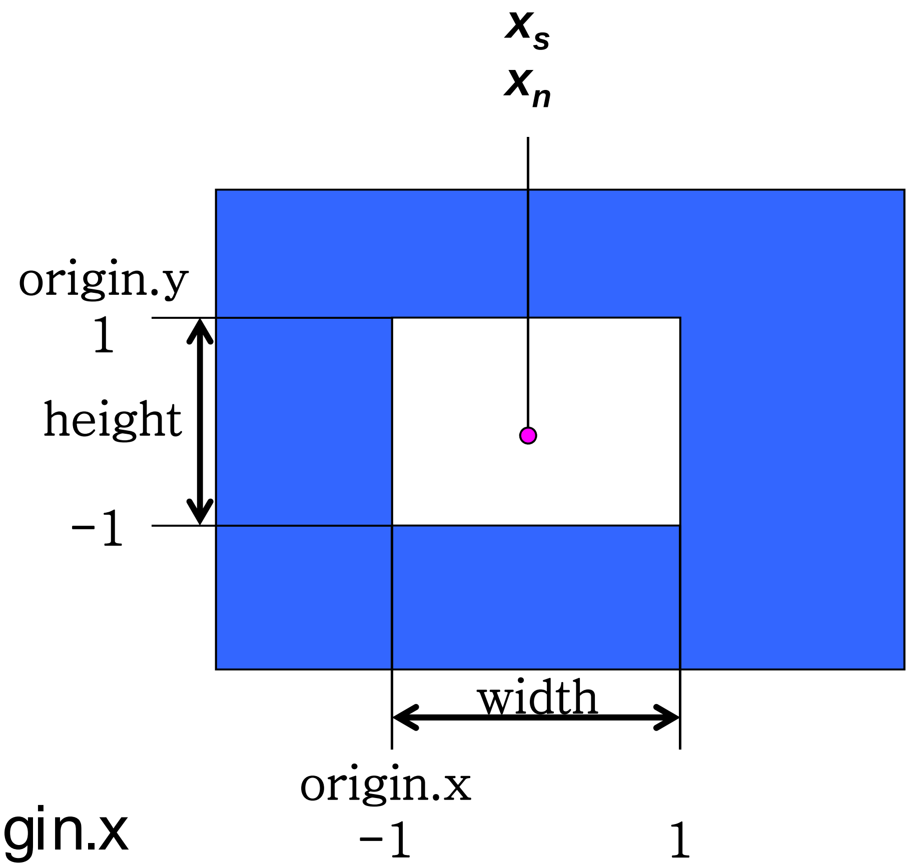
$$\frac{x_s - \text{origin.x}}{\text{width}} = \frac{x_n - (-1)}{1 - (-1)}$$

- **Solve for x_s**

$$x_s = \text{width}\frac{x_n + 1}{2} + \text{origin.x}$$

$$x_s = Ax_n + B$$

$$A = \frac{\text{width}}{2}; \quad B = \frac{\text{width}}{2} + \text{origin.x}$$



Class Objectives were:

- Understand different spaces and basic OpenGL commands
- Understand a continuous world, Julia sets

Any Questions?

- Come up with one question on what we have discussed in the class and submit at the end of the class
  - 1 for already answered questions
  - 2 for typical questions
  - 3 for questions with thoughts or that surprised me
- Submit four times during the whole semester

Homework

- **Go over the next lecture slides before the class**
- **Watch 2 SIGGRAPH videos and submit your summaries before every Tue. class**
  - **Send an email to cs380ta@gmail.com**
  - **Just one paragraph for each summary**

Example:

Title: XXX XXXX XXXX

Abstract: this video is about accelerating the performance of ray tracing. To achieve its goal, they design a new technique for reordering rays, since by doing so, they can improve the ray coherence and thus improve the overall performance.

Homework for Next Class

- Read Chapter 1, Introduction
  - Read "Numerical issues" carefully

Next Time

- **Basic OpenGL program structure and how OpenGL supports different spaces**