CS380: Computer Graphics
Screen Space & World Space

Sung-Eui Yoon
(윤성의)

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

KAIST

Class Objectives

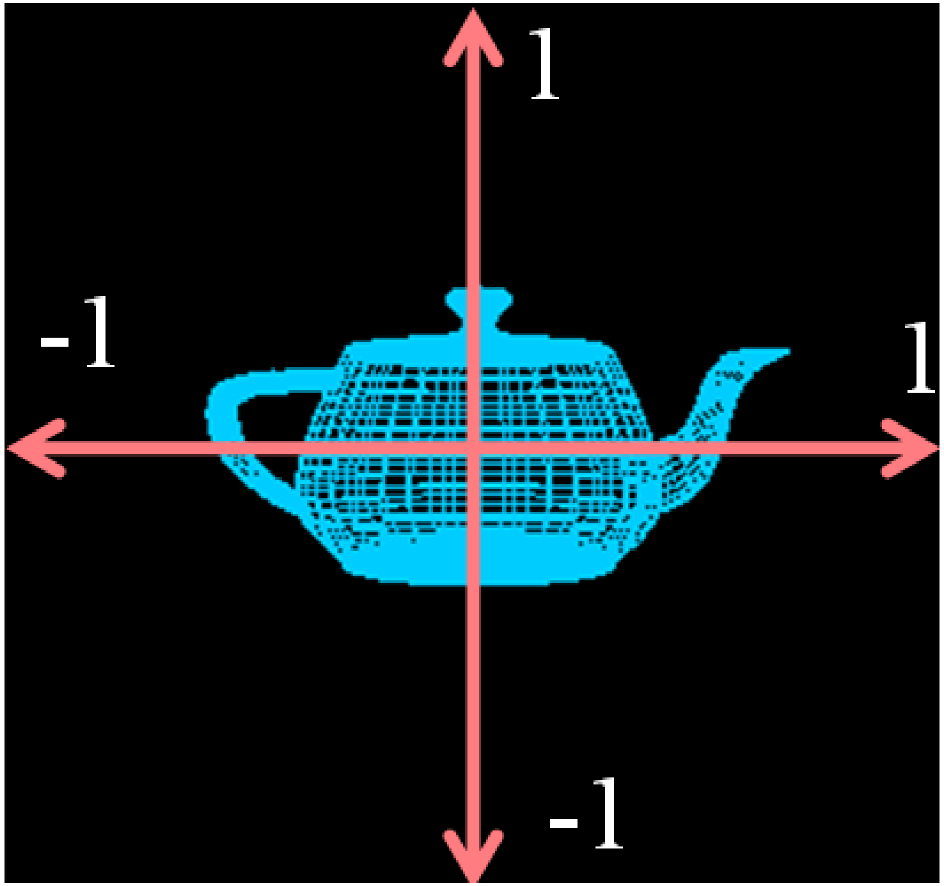
- **Understand different spaces and basic OpenGL commands**
- **Understand a continuous world, Julia sets**

- **Review of prior class:**
  - **Student activities (6~7 programming assignments, paper/video summary submission every week, poster presentation, etc.)**
  - **Grading policy**

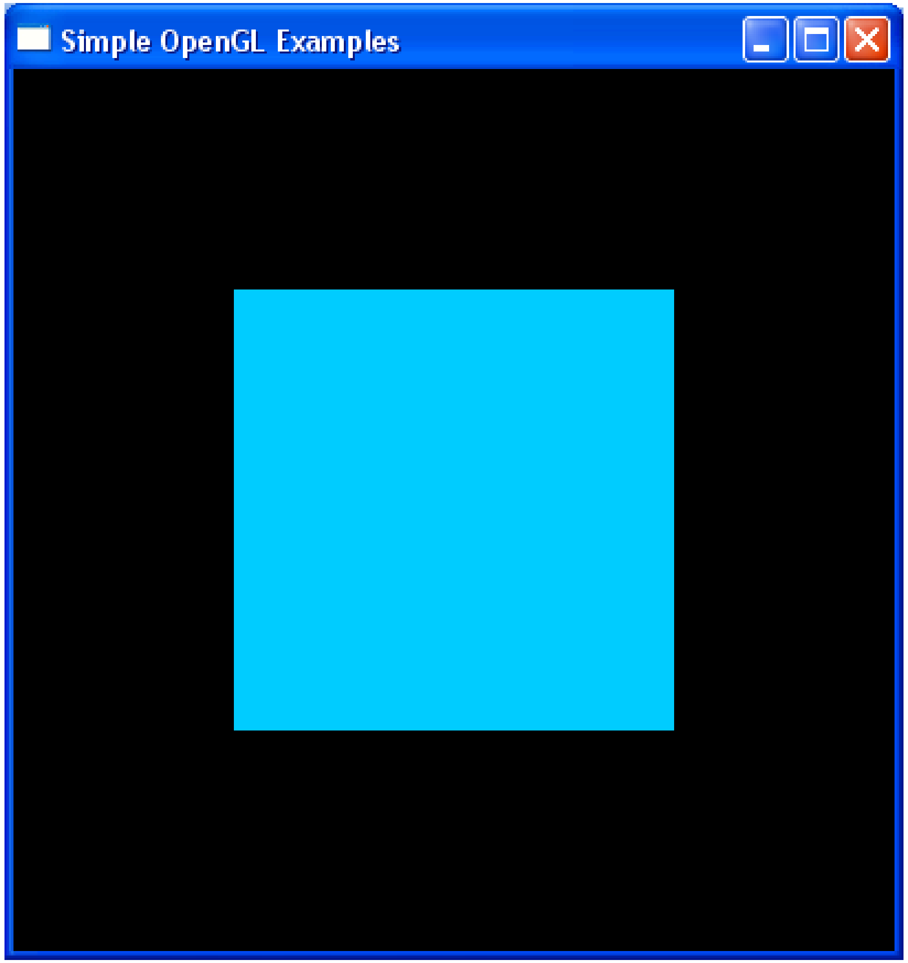
KAIST

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 (Immediate Mode)



Legacy 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

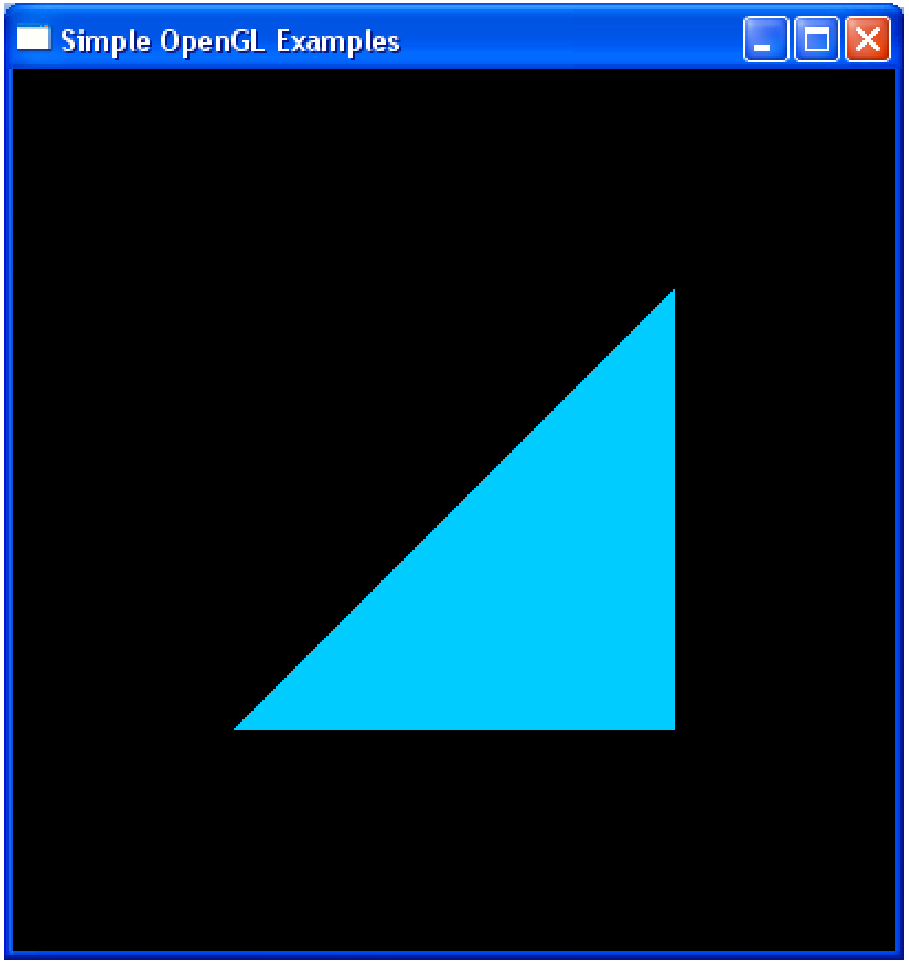
OpenGL Command Syntax

- 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
  - Buffers can be cached in GPU

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()
```


KAIST

Drawing Primitives in OpenGL

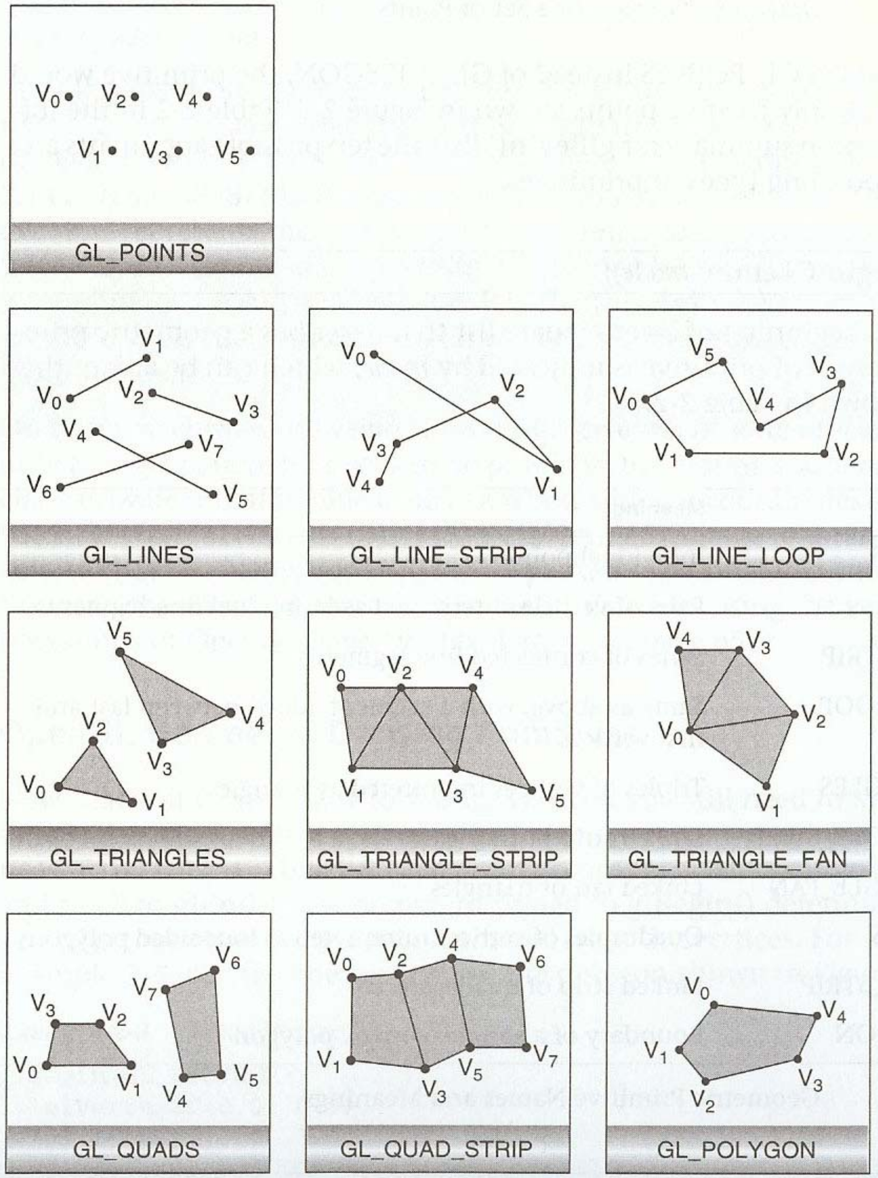
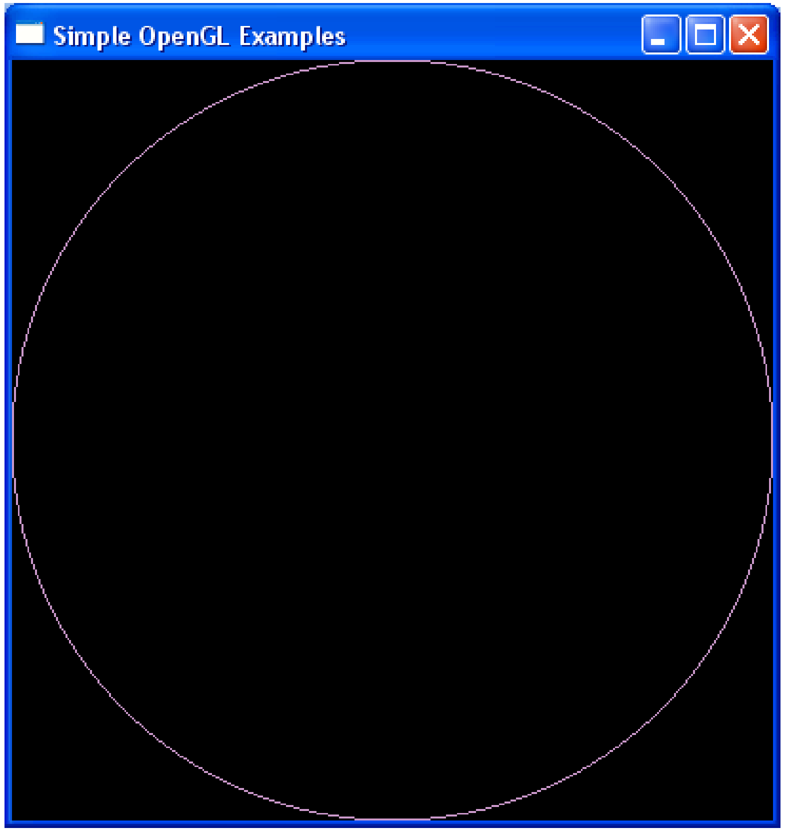


Figure 2-7 Geometric Primitive Types

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

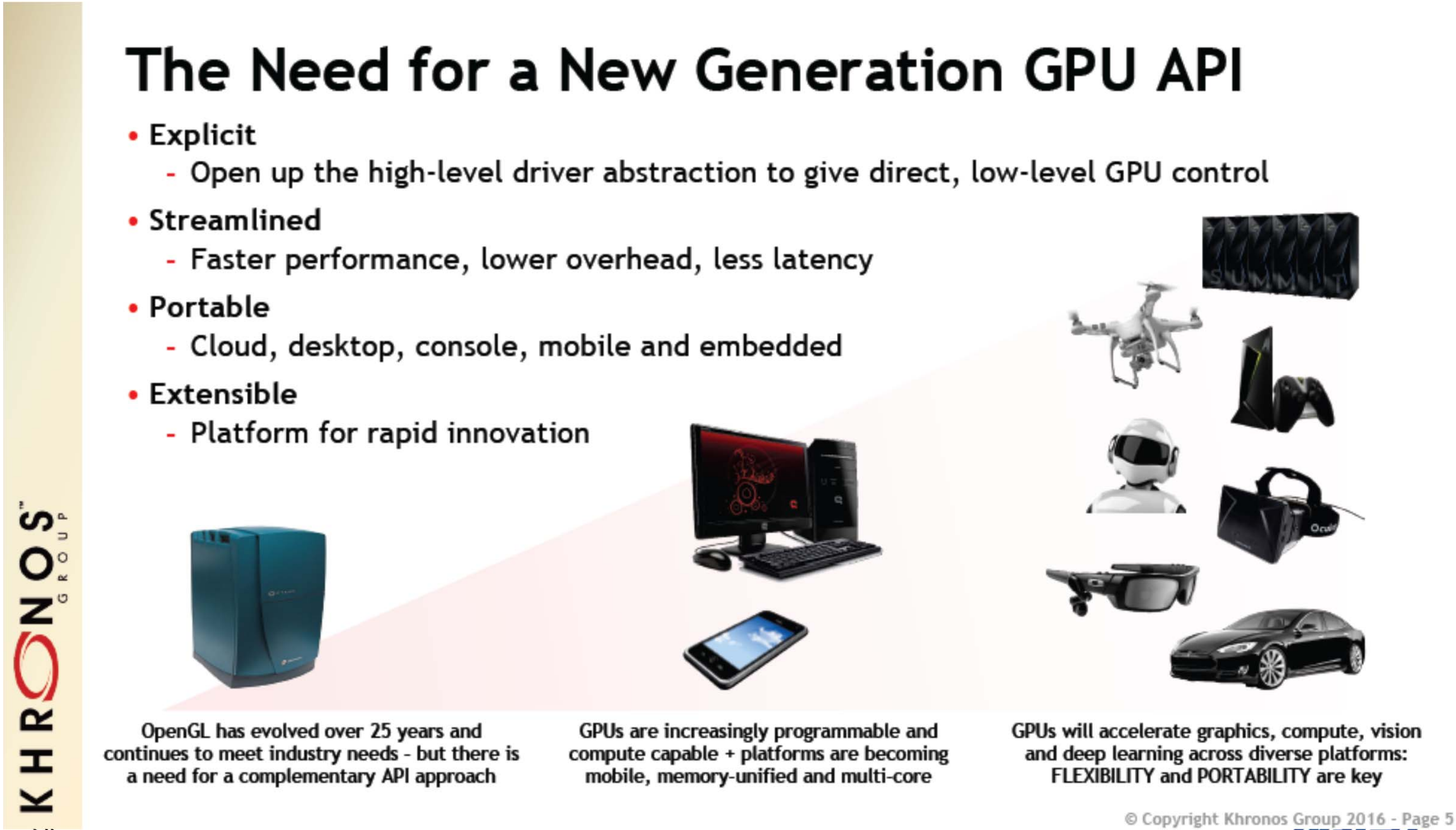
```
#define CheckError(s)                                    \
  {                                                      \
    GLenum error = glGetError();                         \
    if (error)                                           \
      printf("%s in %s\n",gluErrorString(error),s);      \
 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

```
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)

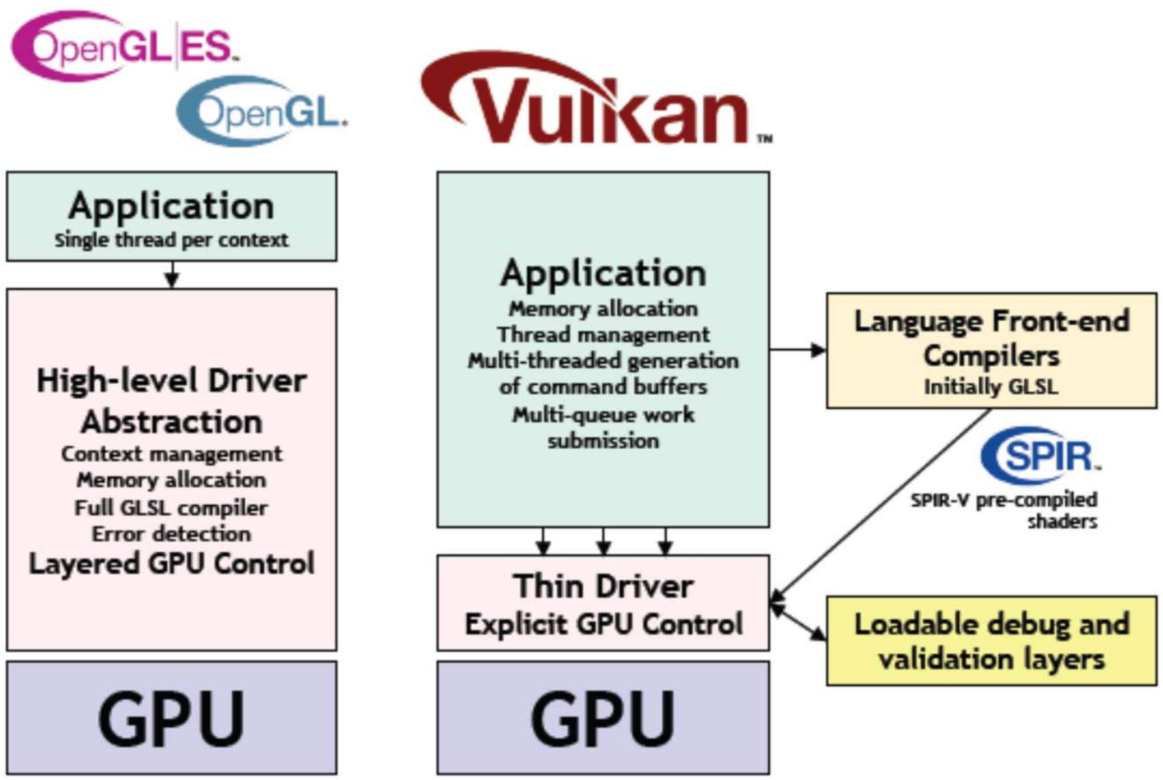
```
#include <iostream>
using namespace std;
#include "vgl.h"
#include "LoadShaders.h"
enum VAO_IDs { Triangles, NumVAOs };
enum Buffer_IDs { ArrayBuffer, NumBuffers };
enum Attrib_IDs { vPosition = 0 };
GLuint VAOs[NumVAOs];
GLuint Buffers[NumBuffers];
const GLuint NumVertices = 6;

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();
}
```

Vulkan: A Recent Change



Benefits of Vulkan

Vulkan Explicit GPU Control



Vulkan 1.0 provides access to
OpenGL ES 3.1 / OpenGL 4.X-class GPU functionality
but with increased performance and flexibility

Vulkan Benefits

Simpler drivers:
Improved efficiency/performance
Reduced CPU bottlenecks
Lower latency
Increased portability

Resource management in app code:
Less hitches and surprises

Command Buffers:
Command creation can be multi-threaded
Multiple CPU cores increase performance

Graphics, compute and DMA queues:
Work dispatch flexibility

SPIR-V Pre-compiled Shaders:
No front-end compiler in driver
Future shading language flexibility

Loadable Layers
No error handling overhead in production code

Educational Issue on CG SWs

- **Recent trends of real-time rendering add additional complexity and lower level details for higher performance**
  - **Away from easy entrance to its field; i.e., not good for educational purposes**

- **Physically-based rendering is getting more widely used**
  - **Understanding principled concepts is more important than fast performance**

Pixar, good dinosaur

KAIST

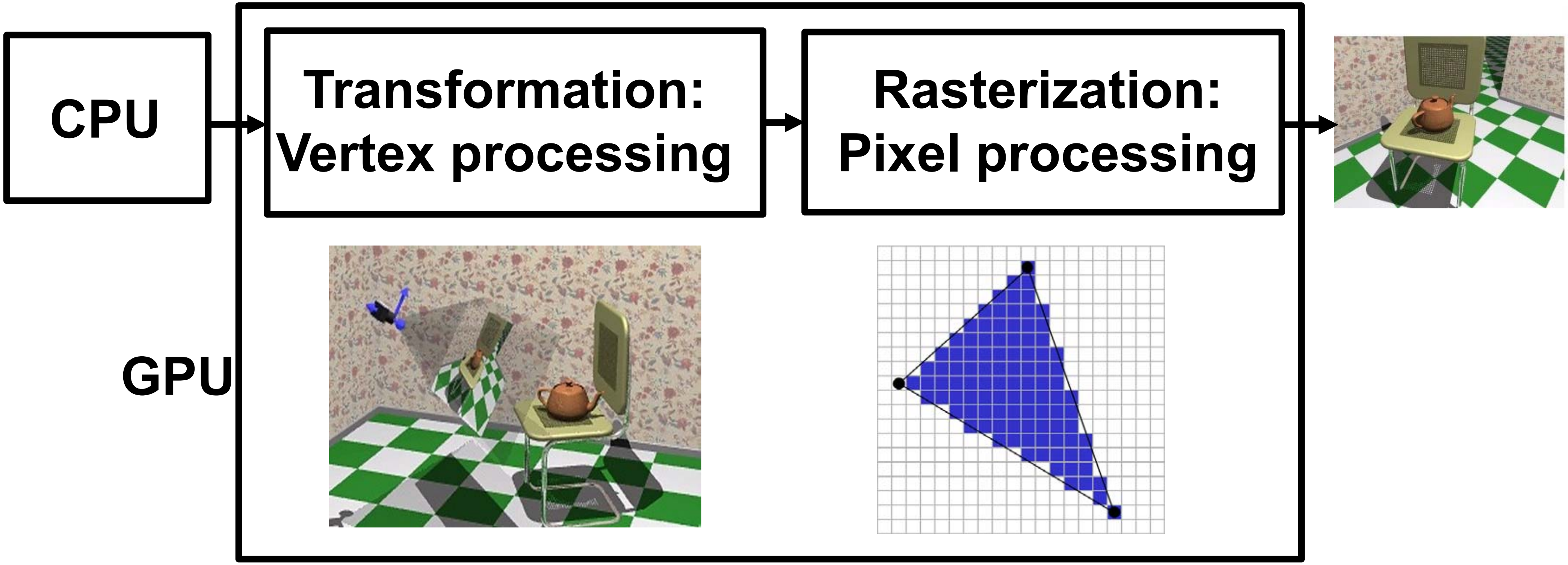
My Approach

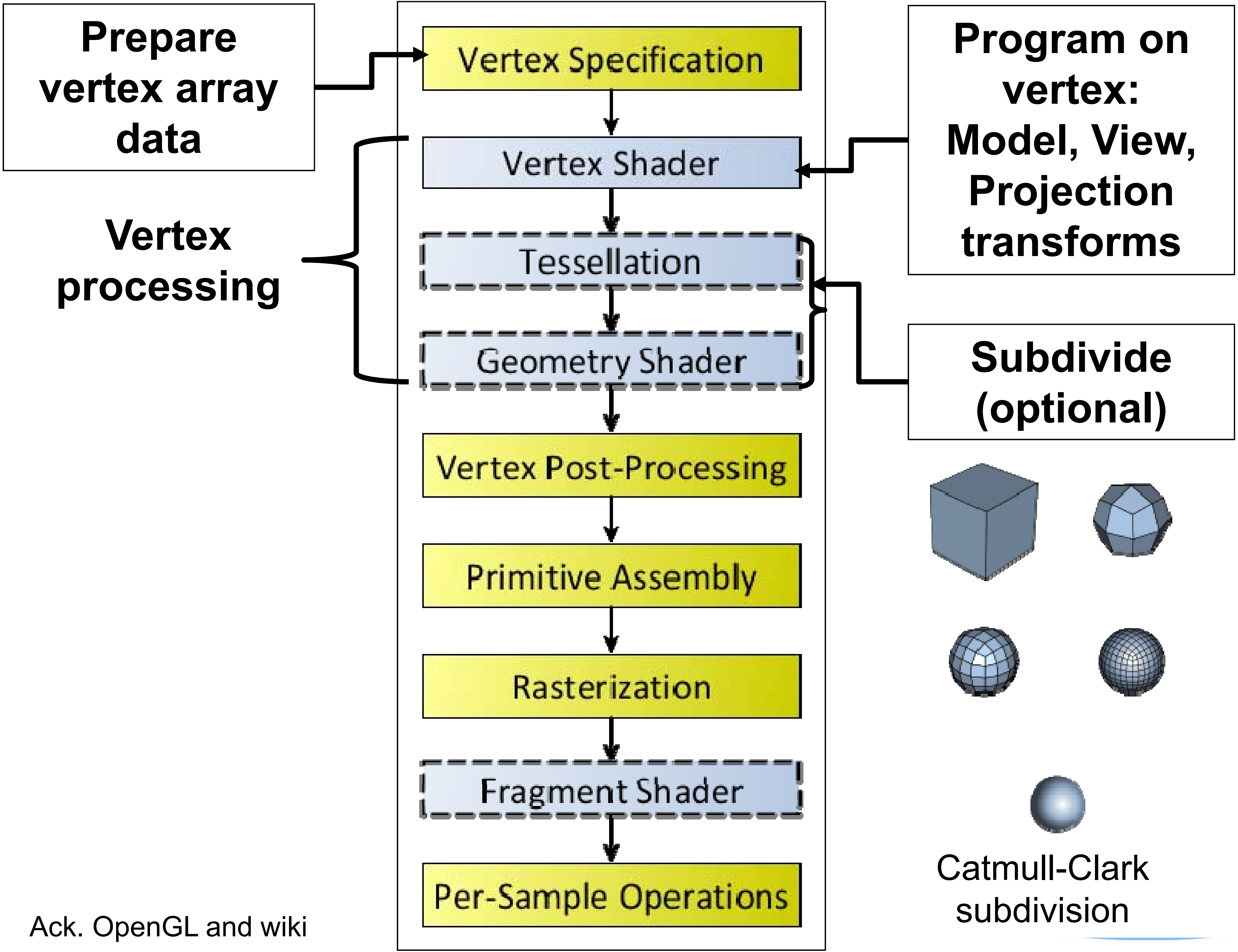
- Focus on fundamental concepts that will last in many coming years
- Use the legacy OpenGL version as a basic teaching tool, thanks to its simplicity
  - Discuss its current form too, to differentiate old and new versions
  - Point out the nature of rapid evolution of computer graphics and computer science

- Programming assignments
  - Based on the legacy OpenGL, which is covered in the class and lab

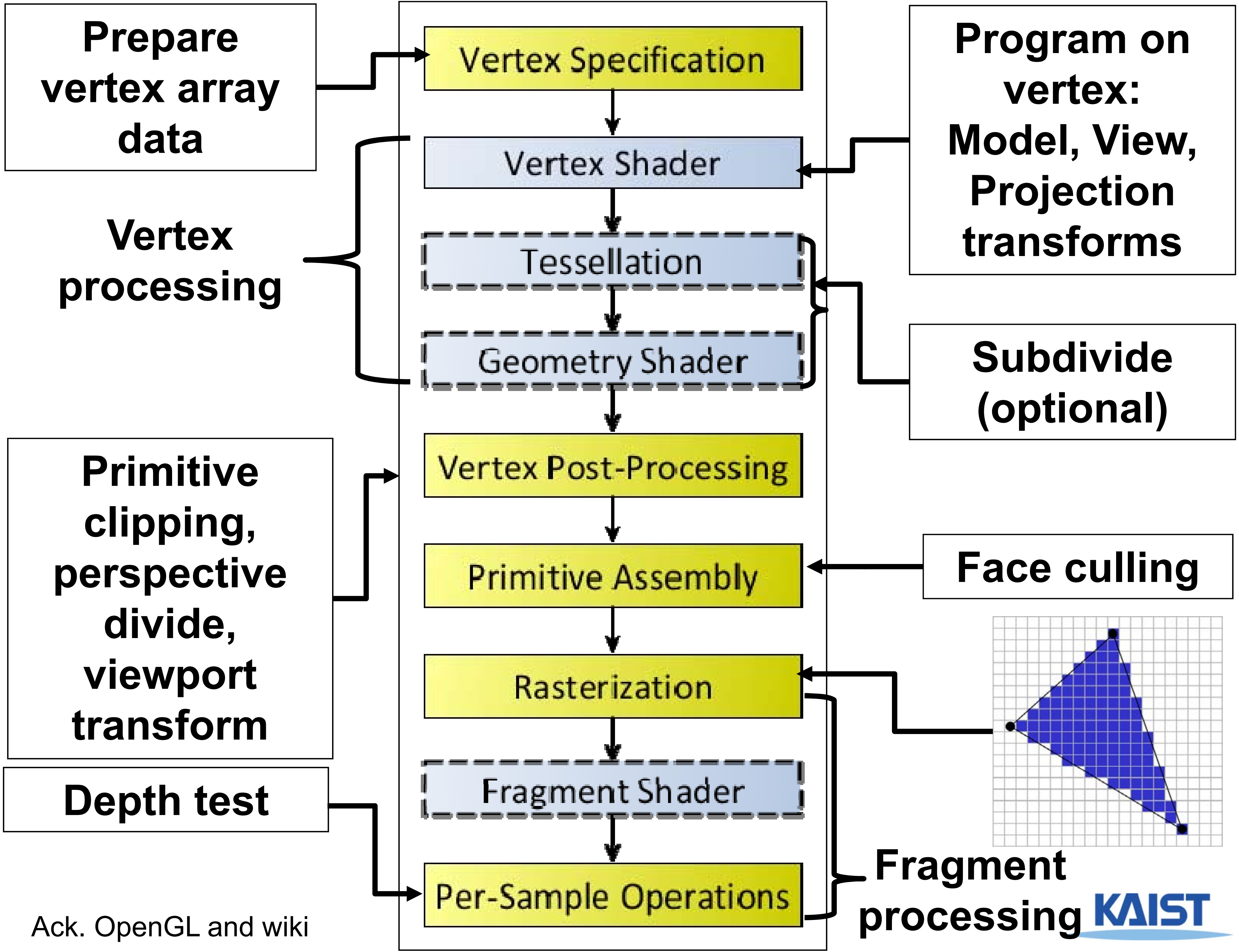
Classic Rendering Pipeline

- **Implemented in various SWs and HWs**





Ack. OpenGL and wiki

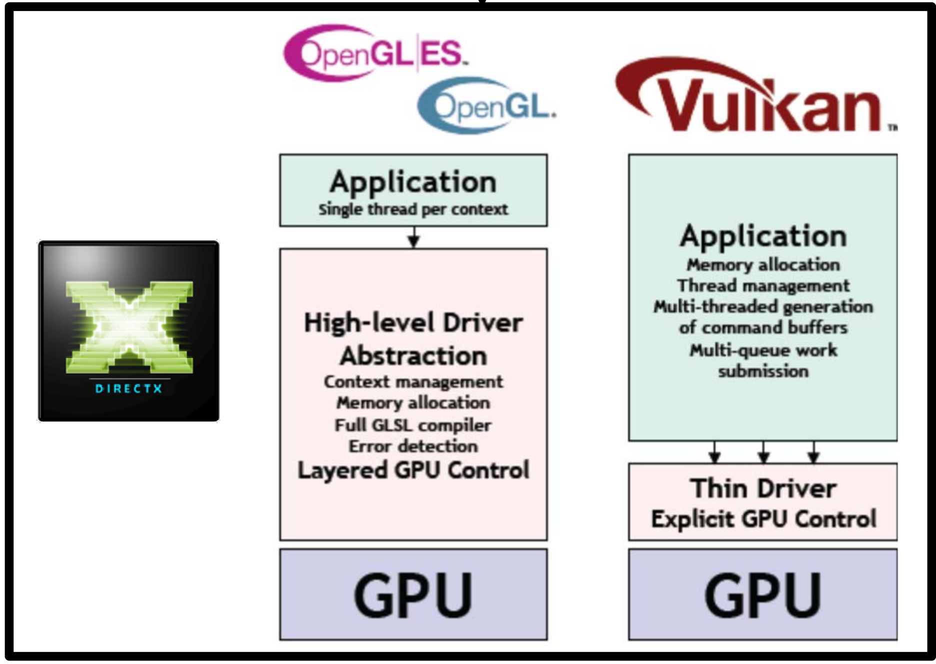


Ack. OpenGL and wiki

Relation to Other CG related Tools/Languages

Game/rendering engine & modeling/animation tools

GPGPU (General-Purpose computing on Graphics Processing Units)



Shading languages (GLSL, HLSL for DirectX)

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**

KAIST

Julia Set - Definition

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

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

Complex numbers: consists of 2 tuples (Real, Imaginary)

E.g., c = a + bi

Various operations

$c_1 + c_2 = (a_1 + a_2) + (b_1 + b_2)i$

$c_1 \cdot c_2 = (a_1a_2 - b_1b_2) + (a_1b_2 + a_2b_1)i$

$(c_1)^2 = ((a_1)^2 – (b_1)^2) + (2 a_1b_1)i$

$|c| = sqrt(a^2 + b^2)$

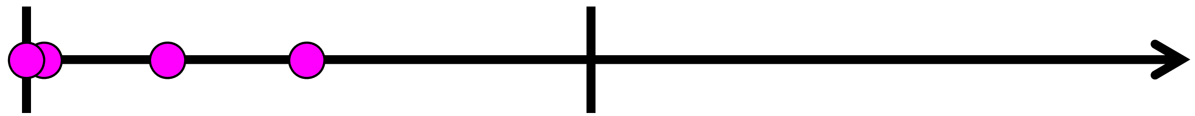
KAIST

Convergence Example

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

How about $x_0 = 0.5$?

$x_{0\text{-}4}$ = 0.5, 0.25, 0.0625, 0.0039

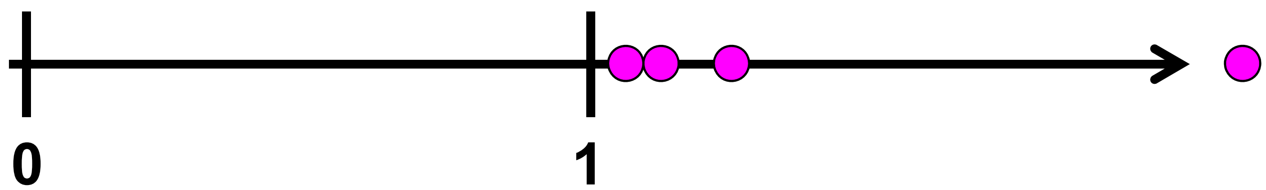


Convergence Example

- **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 $x_0 = 1.1$?

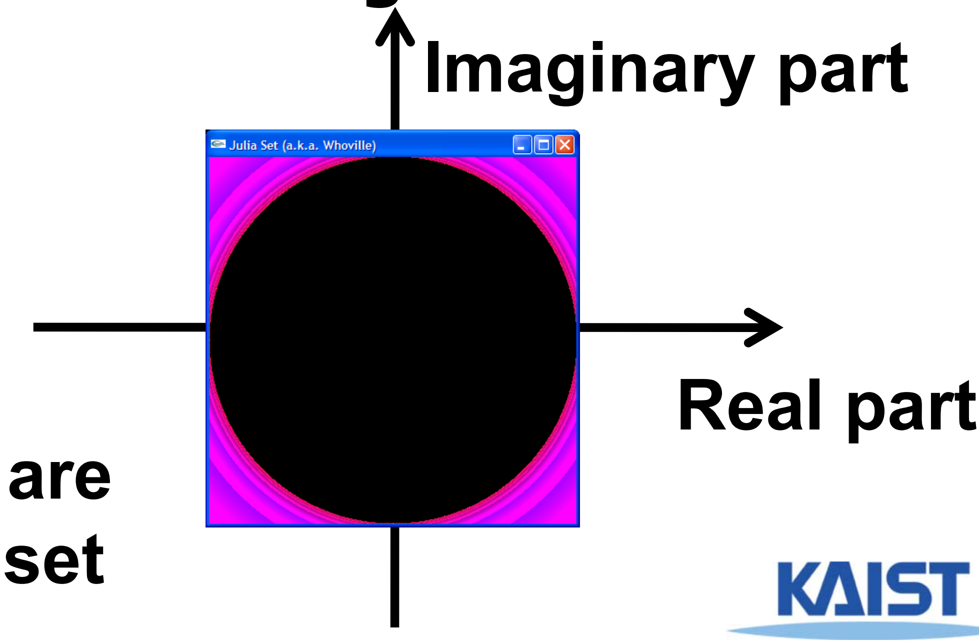
x_{0-4} = 1.1, 1.21, 1.4641, 2.14358



KAIST

Convergence Properties

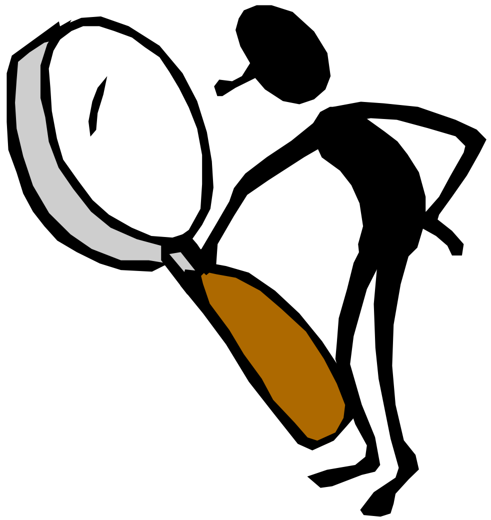
- Suppose c = [0,0], for what complex values of p does the series converge?
- For real numbers:
  - If $|x_i| > 1$, then the series diverges
- For complex numbers
  - If $|p_i| > 2$, then the series diverges
  - Loose bound



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);
}
```



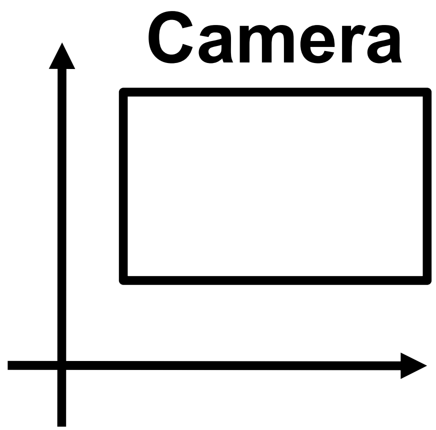
i & r are used to assign a color

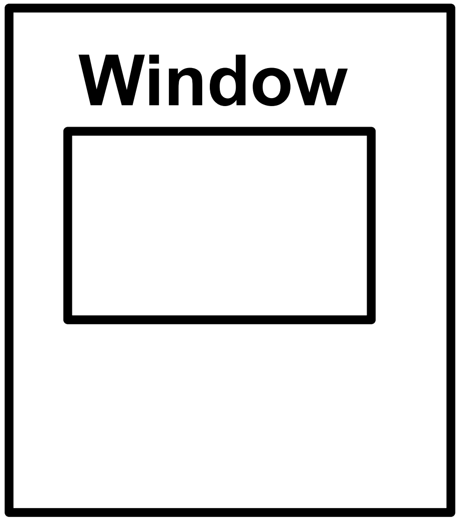
KAIST

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

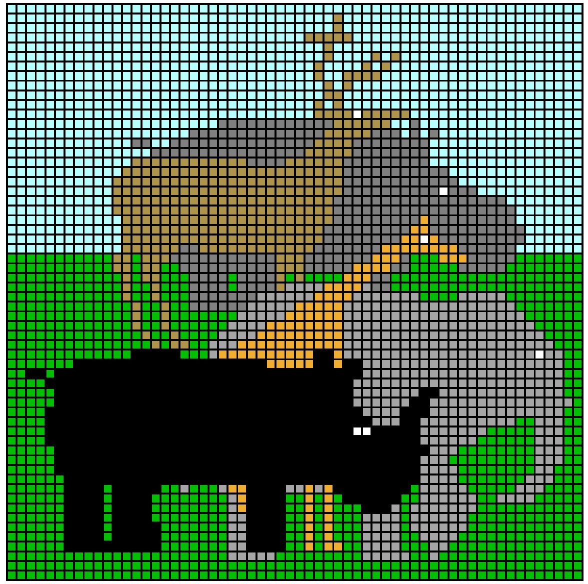




Screen Space

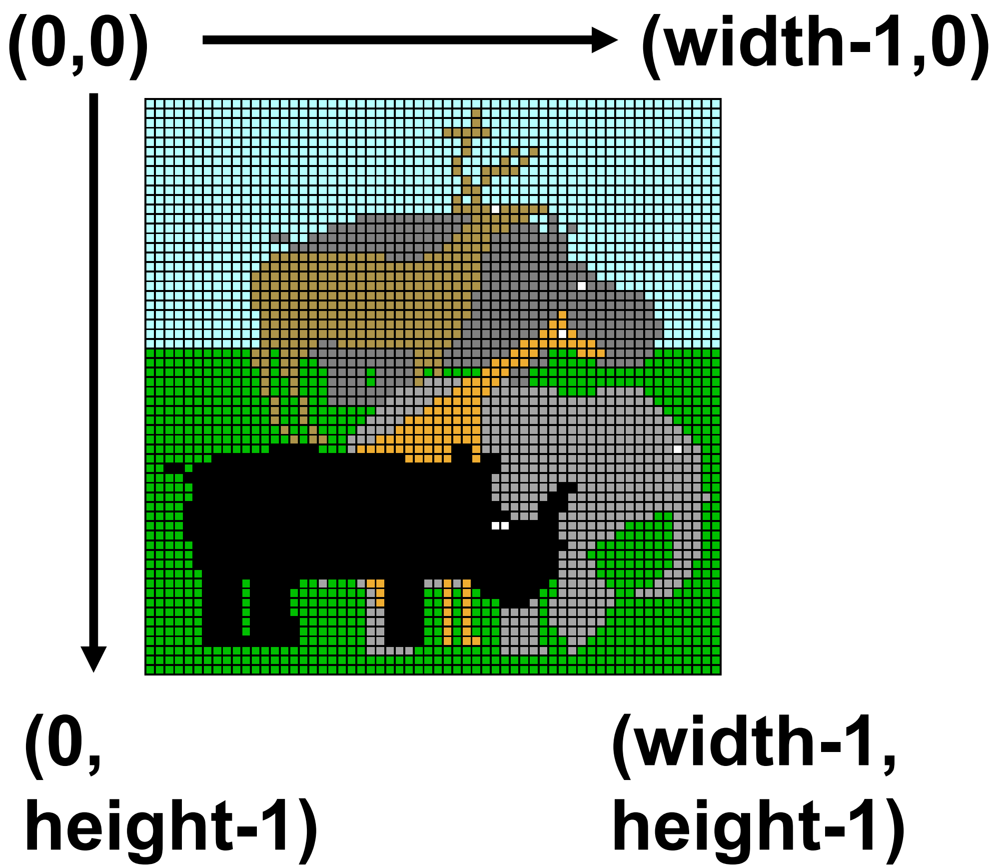
- Graphical image is presented by setting colors for a set of discrete samples called "pixels"
  - Pixels displayed on screen in windows
- Pixels are addressed as 2D arrays
  - Indices are "screen-space" coordinates

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

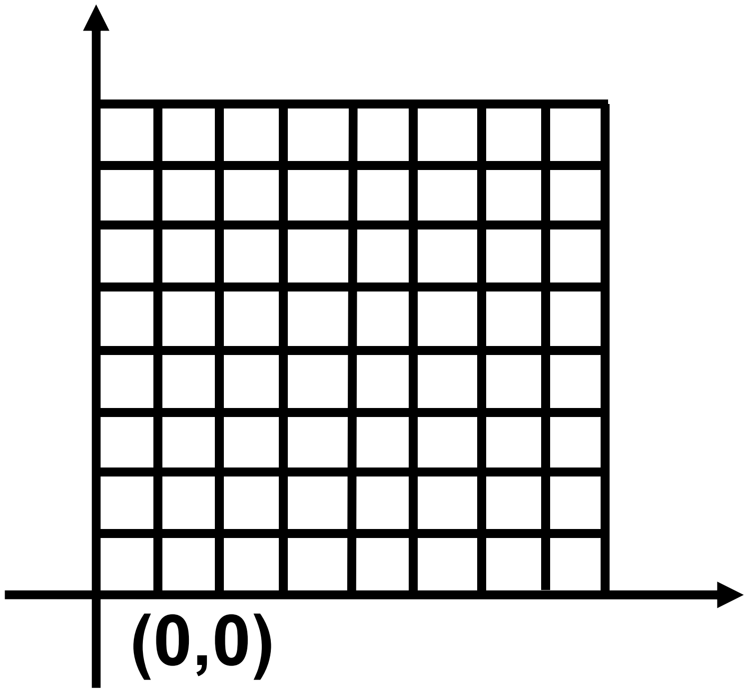


(0,height-1) (width-1, height-1)

Coordinate Conventions



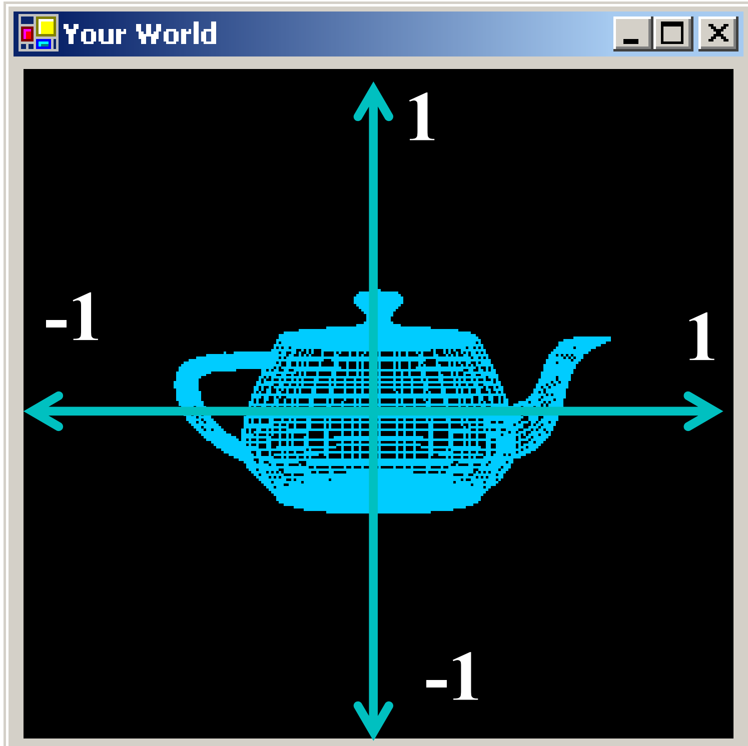
Windows Screen Coordinates



OpenGL Screen Coordinates

Normalized Device Coordinates

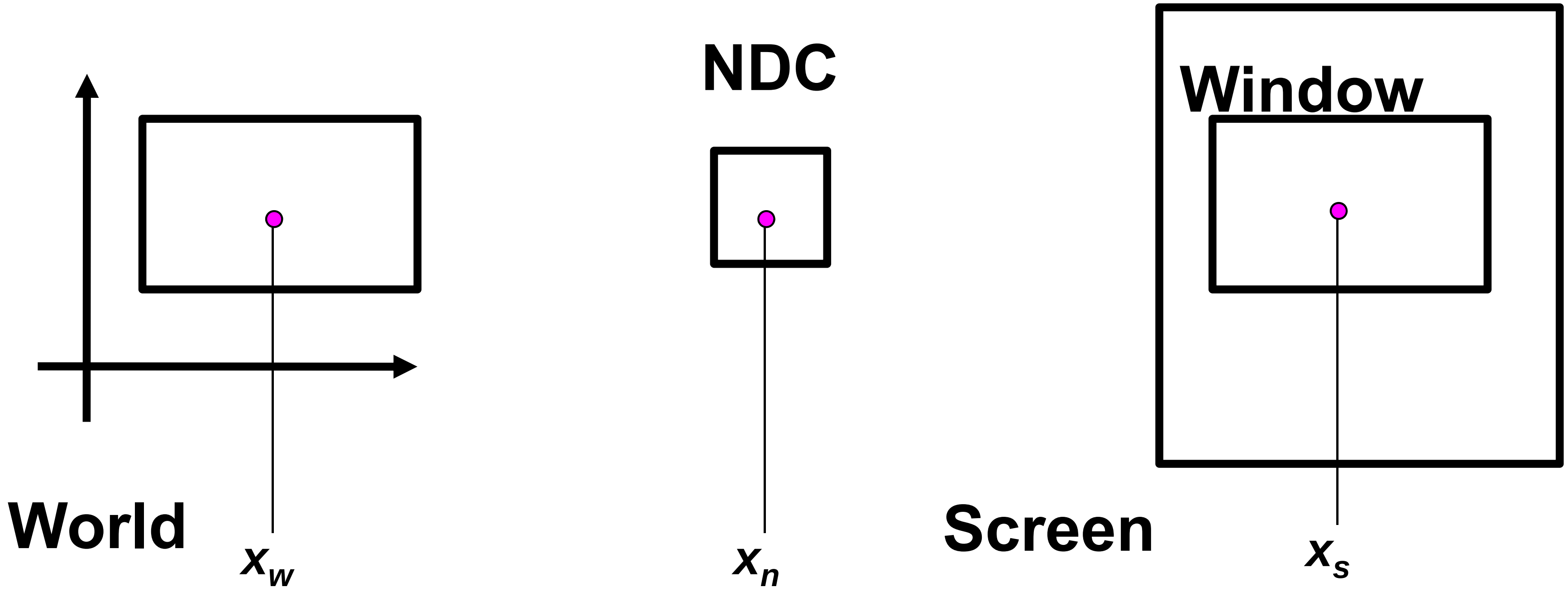
- Intermediate "rendering-space"
  - Compose world and screen space
- Sometimes called "canonical screen space"



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



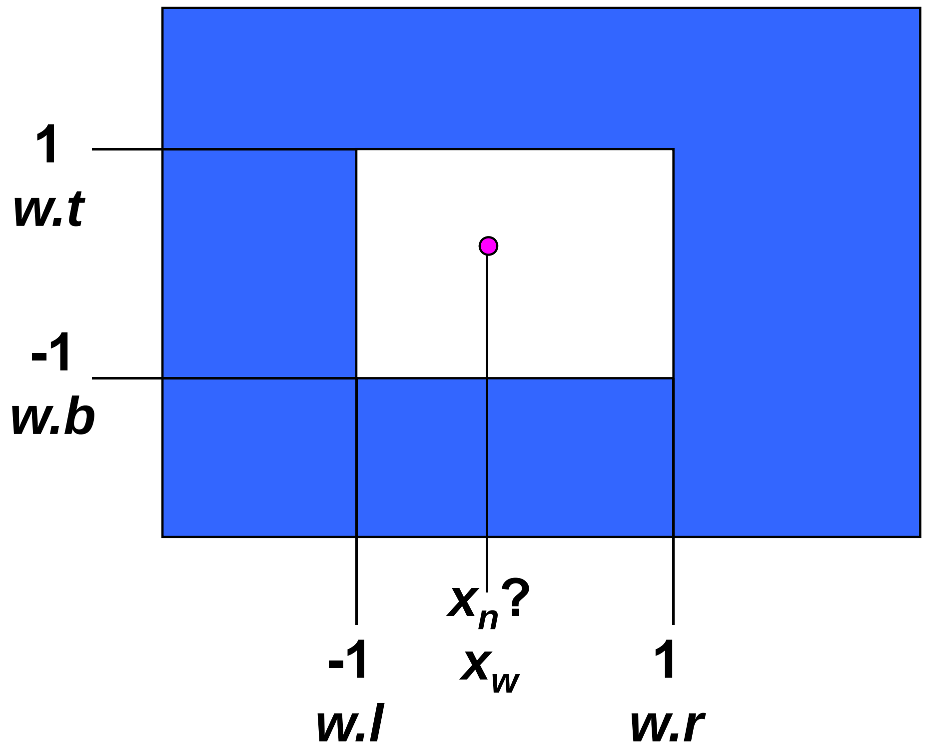
World Space to NDC

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

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

$$x_n = Ax_w + B$$

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



NDC to Screen Space

- **Same approach**

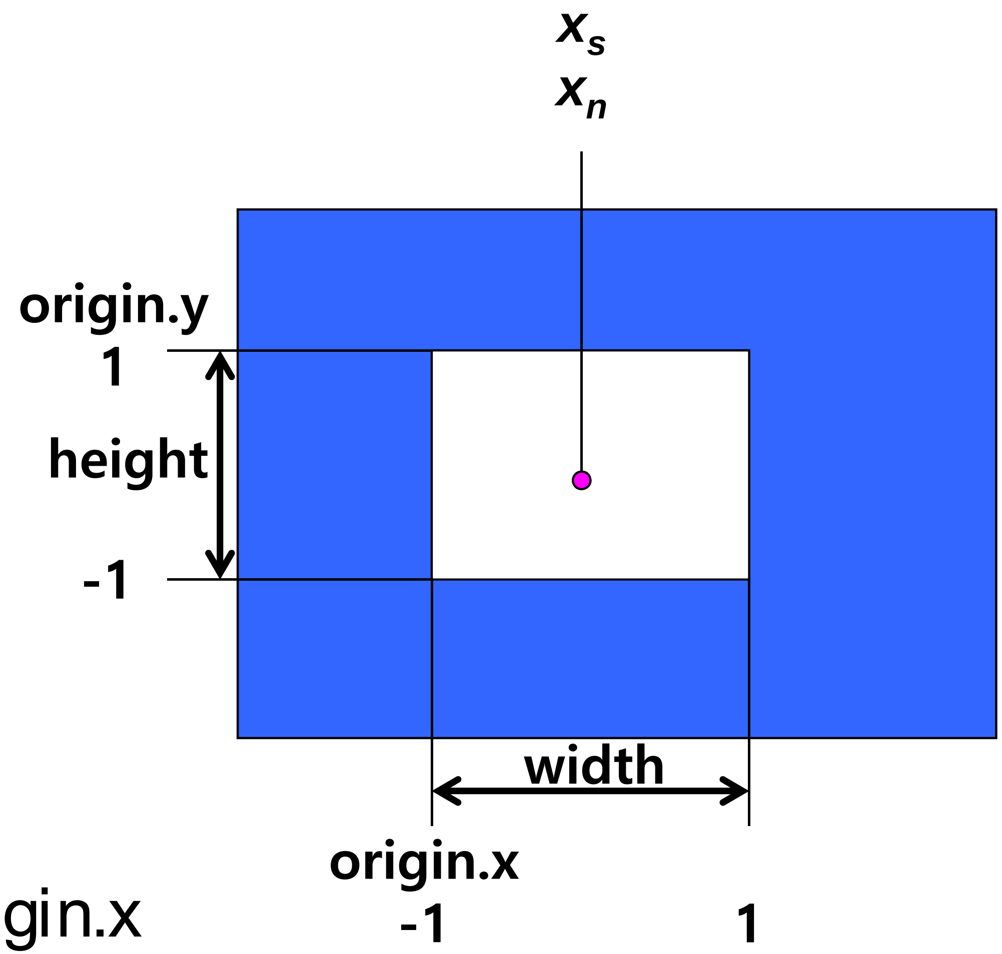
$$\frac{x_s - \text{origin}.x}{\text{width}} = \frac{x_n - (-1)}{1 - (-1)}$$

- **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 questions with thoughts or that surprised me**
- **Submit two times during the whole semester**
  - **Multiple questions in one time are counted as once**

Homework

- **Go over the next lecture slides before the class**
- **Watch 2 SIGGRAPH videos and submit your summaries before every Mon. class**
  - **Submit online through our course homepage**
  - **Just one paragraph for each summary**

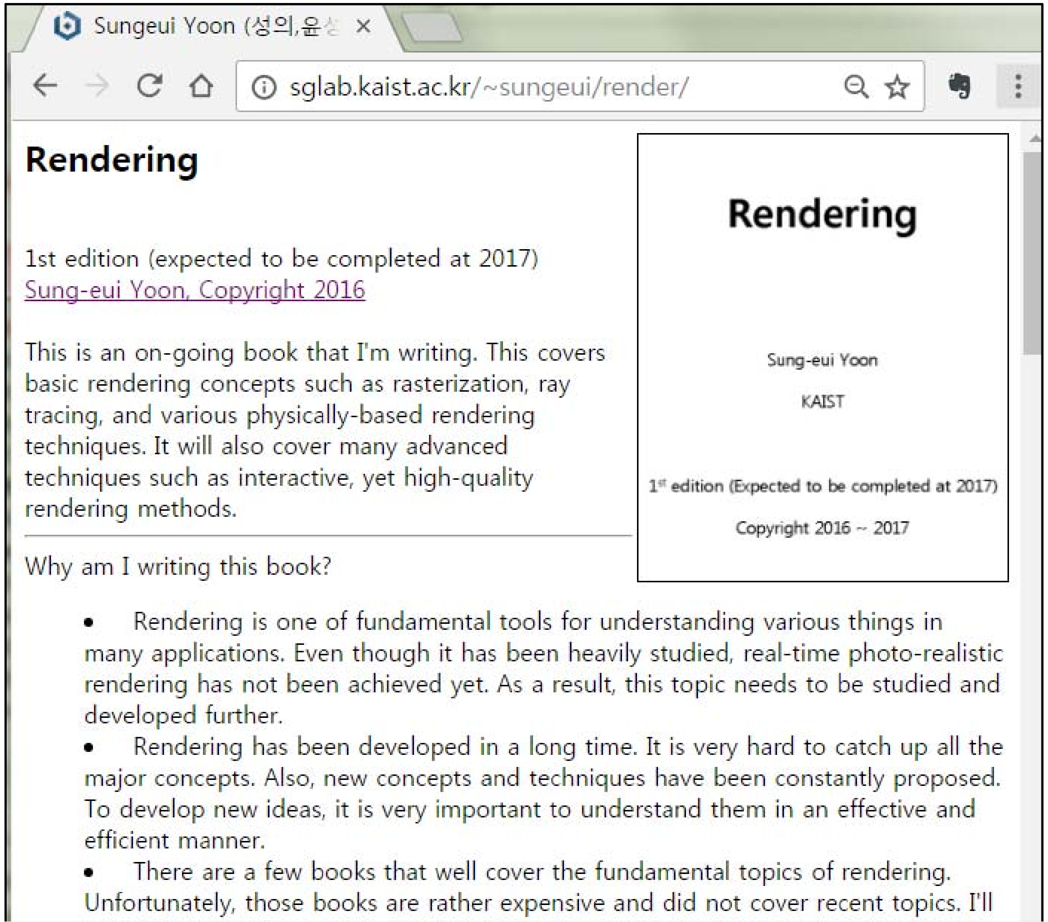
Example: (English or Korean is possible)

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.

KAIST

Homework for Next Class

- **Read:**
  - **Chapter 1, Introduction**
  - **Chapter 2, Classic Rendering pipeline**



Rendering

1st edition (expected to be completed at 2017)
Sung-eui Yoon, Copyright 2016

This is an on-going book that I'm writing. This covers basic rendering concepts such as rasterization, ray tracing, and various physically-based rendering techniques. It will also cover many advanced techniques such as interactive, yet high-quality rendering methods.

Rendering

Sung-eui Yoon

KAIST

1st edition (Expected to be completed at 2017)

Copyright 2016 ~ 2017

Why am I writing this book?

- Rendering is one of fundamental tools for understanding various things in many applications. Even though it has been heavily studied, real-time photo-realistic rendering has not been achieved yet. As a result, this topic needs to be studied and developed further.
- Rendering has been developed in a long time. It is very hard to catch up all the major concepts. Also, new concepts and techniques have been constantly proposed. To develop new ideas, it is very important to understand them in an effective and efficient manner.
- There are a few books that well cover the fundamental topics of rendering. Unfortunately, those books are rather expensive and did not cover recent topics. I'll

Next Time

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