



Introduction

Computer Graphics

Yu-Ting Wu

Outline

- [Introduction to computer graphics](#)
- [Introduction to graphics programming](#)
- [Homework assignments and rendering competition](#)

Outline

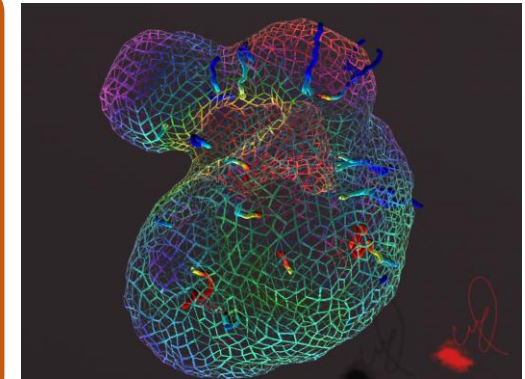
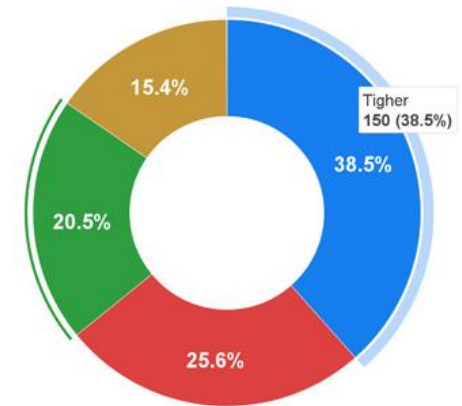
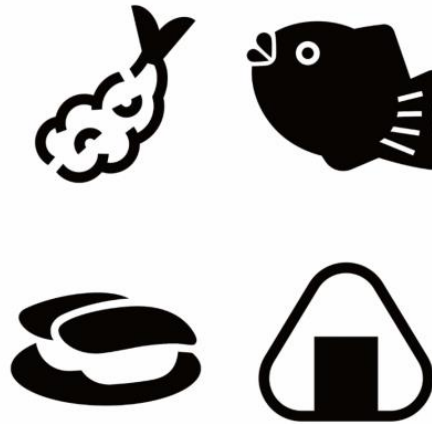
- **Introduction to computer graphics**
- Introduction to graphics programming
- Homework assignments and rendering competition

Overview

What is Computer Graphics

- A sub-field of computer science that studies methods for **digitally synthesizing** and **manipulating** visual content (from *wiki*)
- Is concerned with all aspects of **producing pictures or images using a computer** (from our *textbook*)

These are All Computer Graphics



What we will focus on in this course

Goals of 3D Computer Graphics

- **Digitally synthesize** and **manipulate** a virtual world



Goals of 3D Computer Graphics (cont.)

- **Digitally synthesize** and **manipulate** a virtual world



Copyright © Ralph Breaks the Internet: Wreck-It Ralph 2, 2018, Disney Inc.

Applications of Computer Graphics

Video Games



Copyright © Final Fantasy VII Rebirth, 2024, SQUARE ENIX Inc.

Digital Visual Effects (VFX)



Copyright © Kingdom of the Planet of the Apes, 2024, 20th Century Studios Inc.

Digital Visual Effects (VFX) (cont.)



Copyright © Godzilla Minus One, 2023, TOHO, Inc.

Featured Animations



Copyright © Inside Out 2, 2024, Disney Inc.

Anime



Copyright © 諫山創・講談社／「進撃の巨人」製作委員会

Extended Reality (XR: VR/AR/MR)

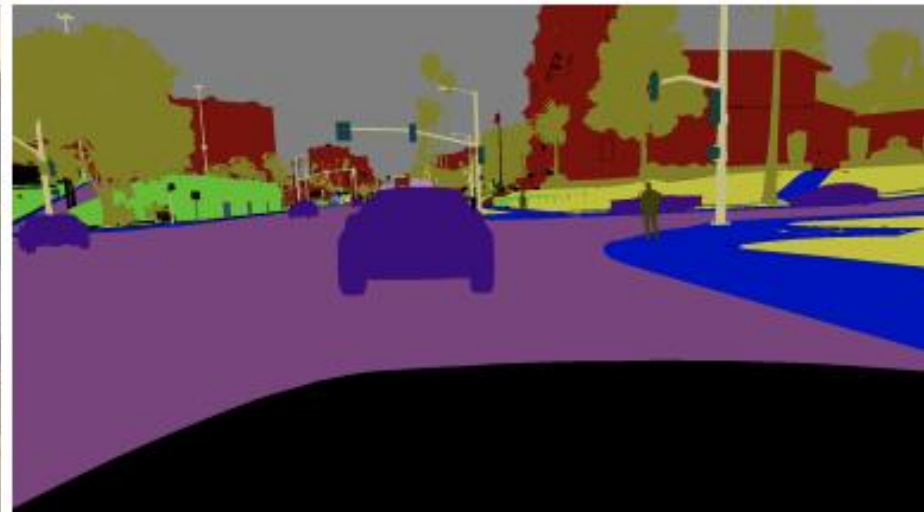
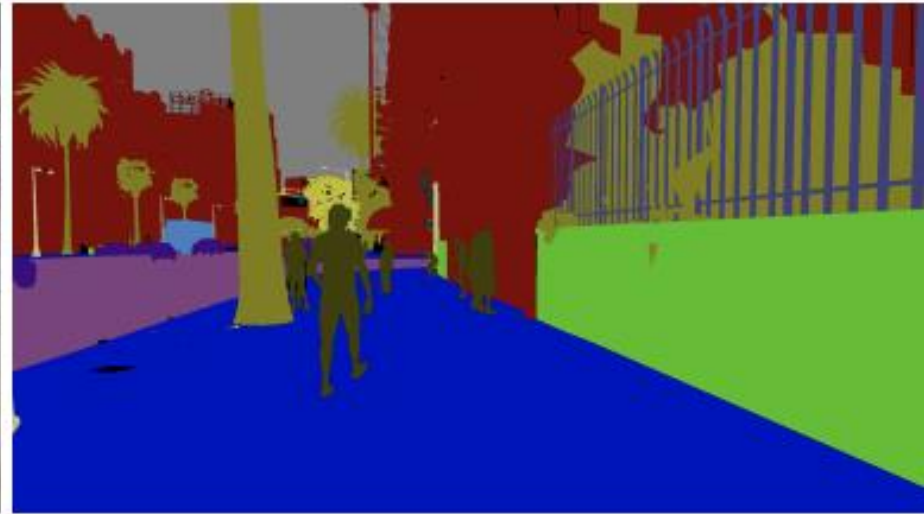


Computer-Aided Design



Machine (Deep) Learning

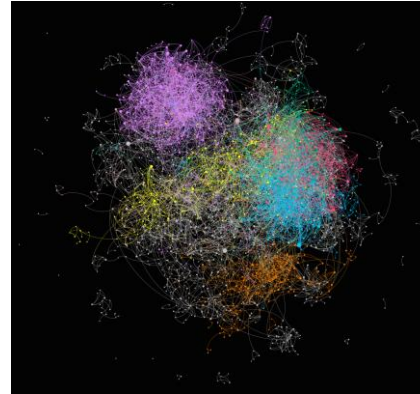
GTA5 Database



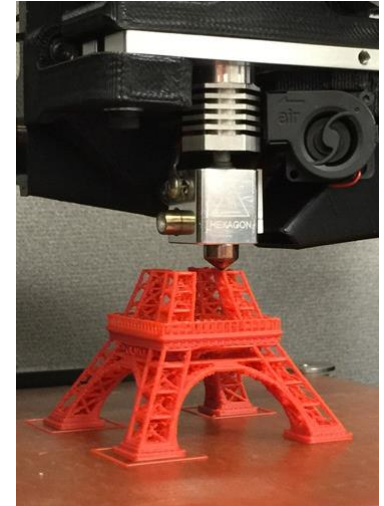
More Applications



Simulation



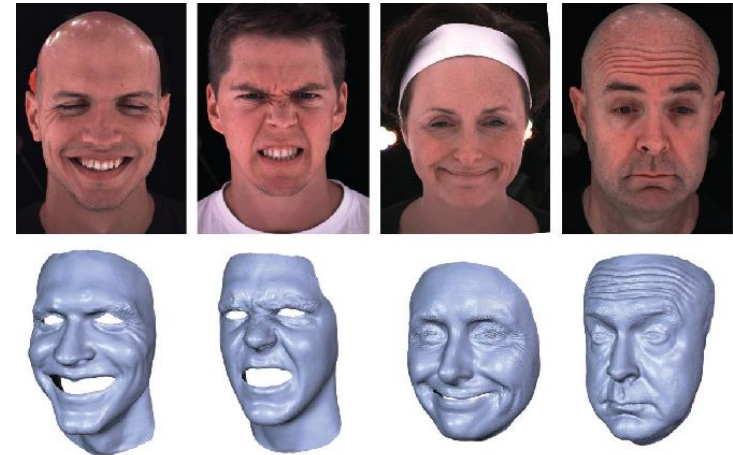
Data Vis



Fabrication



Medical Imaging



3D Reconstruction

A Quick Overview for How to Synthesize an Image

How to Synthesize an Image

- Model geometry of the 3D objects (scene)



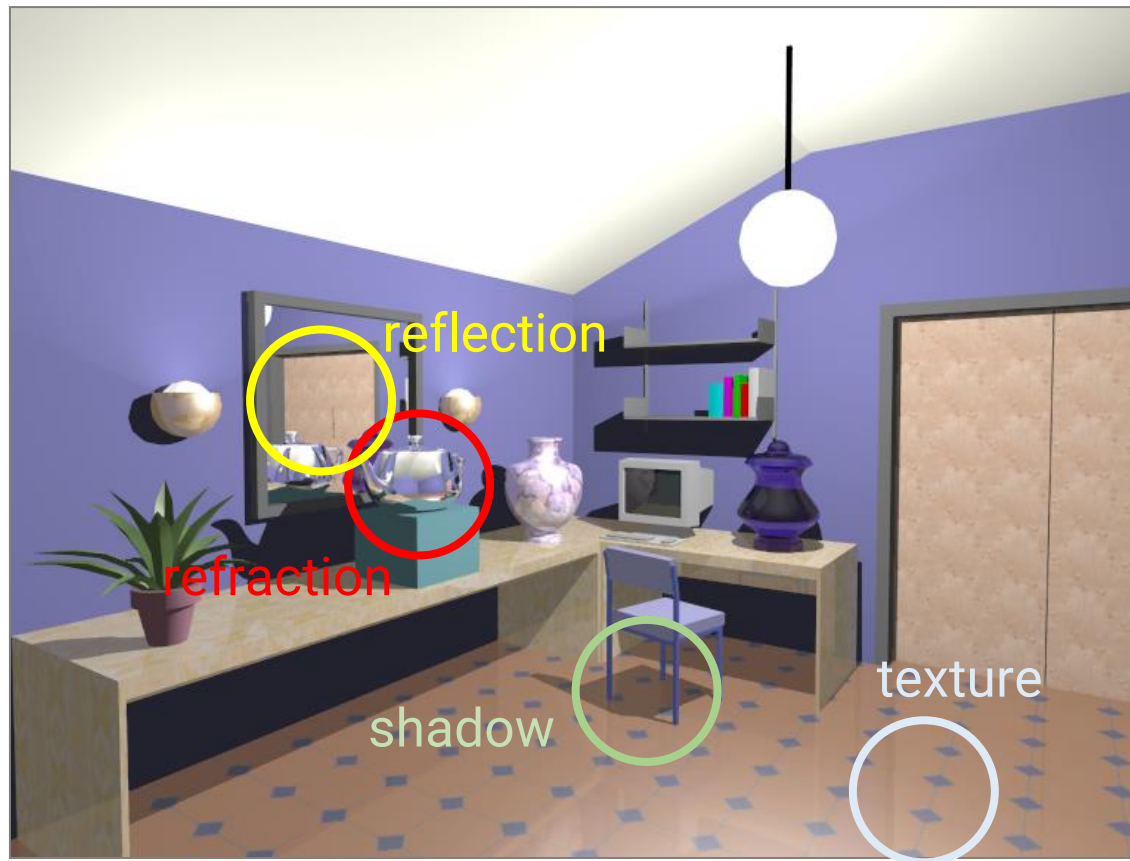
How to Synthesize an Image (cont.)

- Model materials of the 3D objects and simulate lighting



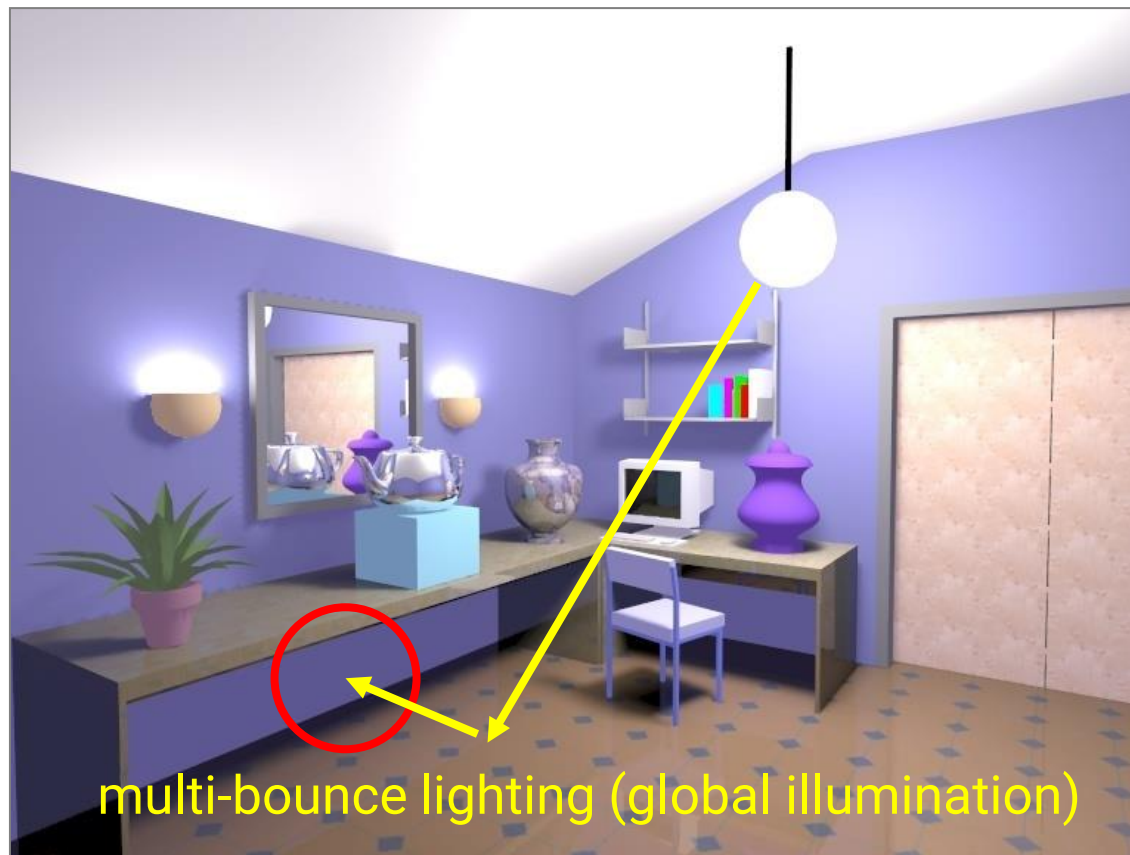
How to Synthesize an Image (cont.)

- Simulate more realistic materials and lighting phenomena



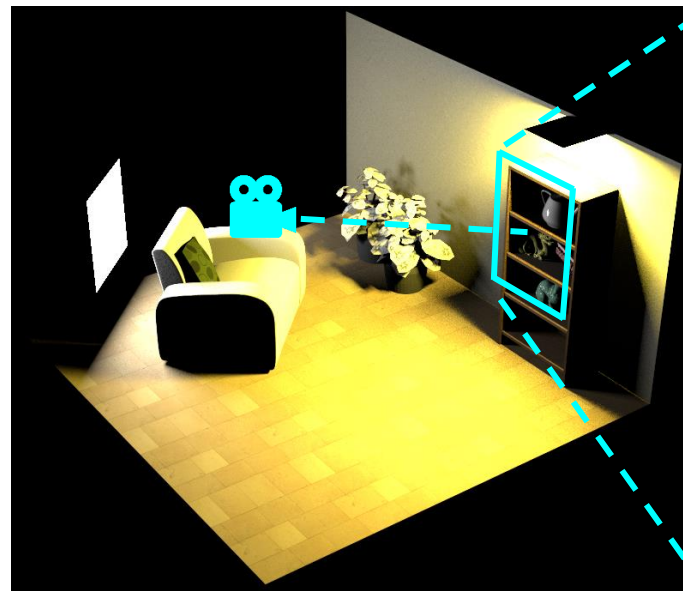
How to Synthesize an Image (cont.)

- Simulate more complex light paths



How to Synthesize an Image (cont.)

- Most displays are 2D, so we need to generate images from the 3D world
- Just like taking a picture with a camera in our daily lives
 - But with a **virtual camera** and a **virtual film**



3D virtual world



rendered image

How to Synthesize an Image (cont.)



Relevant Fields

- Traditionally, we will categorize *computer graphics*, *computer vision*, and *image processing* by their inputs and outputs:

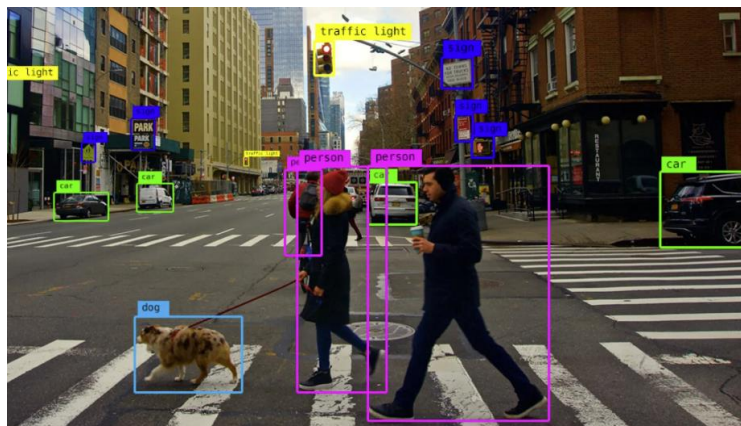
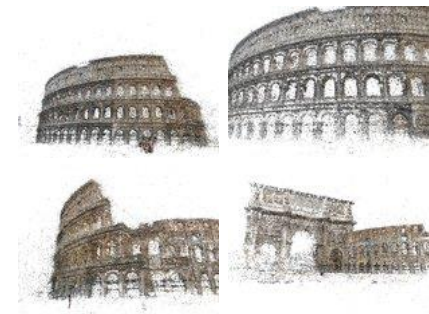
outputs

inputs		descriptions	images
	descriptions		<i>computer graphics</i>
	images	<i>computer vision</i>	<i>Image processing</i>

- However, the gaps are much vaguer now!

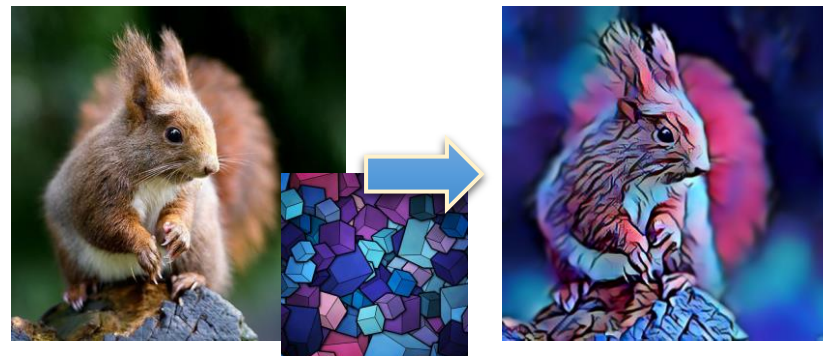
Relevant Fields (cont.)

- **Computer vision**
 - From images to scene descriptions



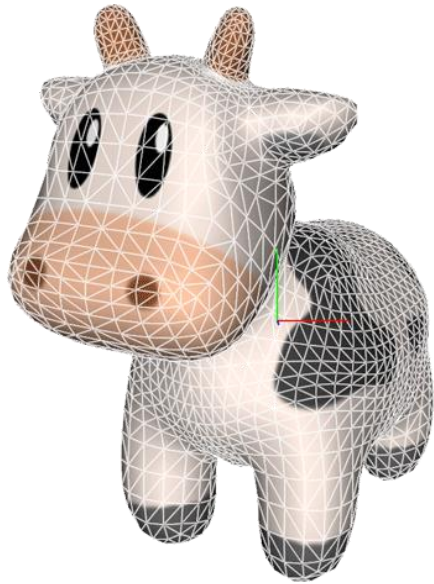
Relevant Fields (cont.)

- **Image processing**
 - From an image to a better image



Major Topics of Computer Graphics

Three Pillars of Computer Graphics



Modeling



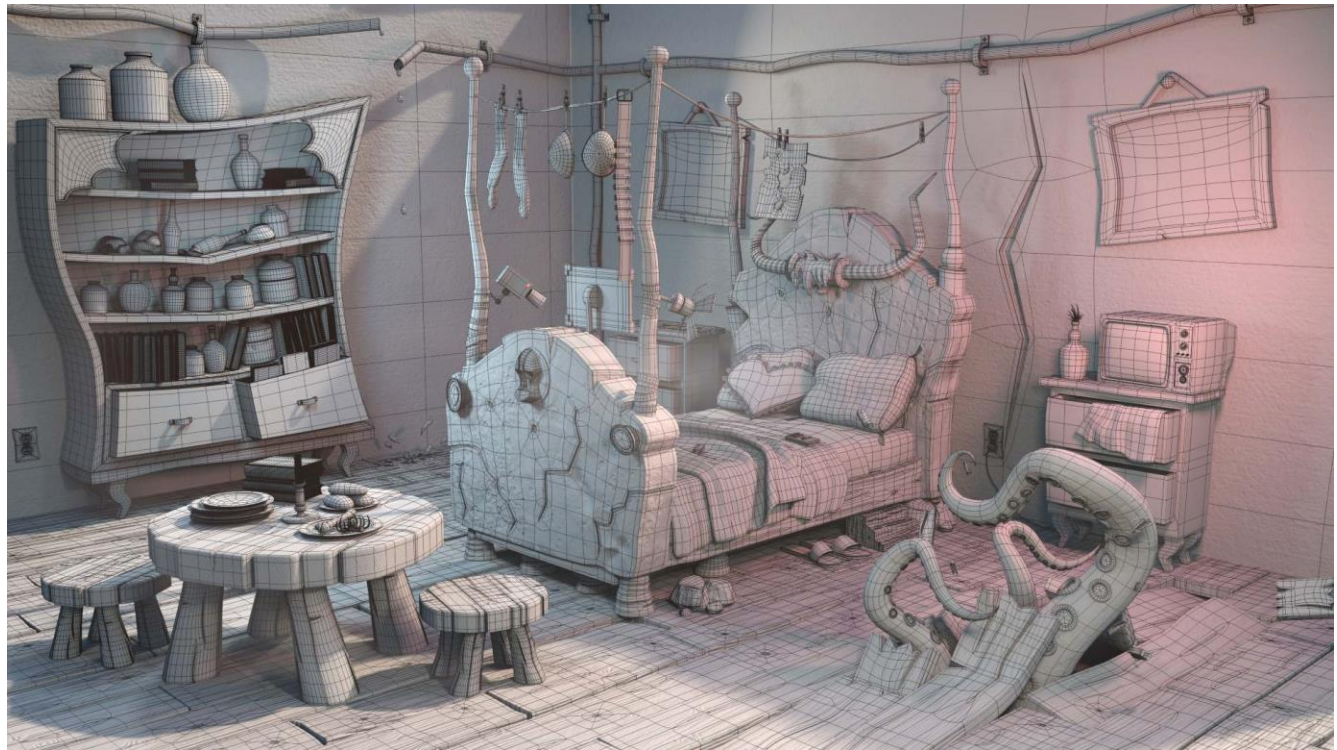
Rendering



Animation

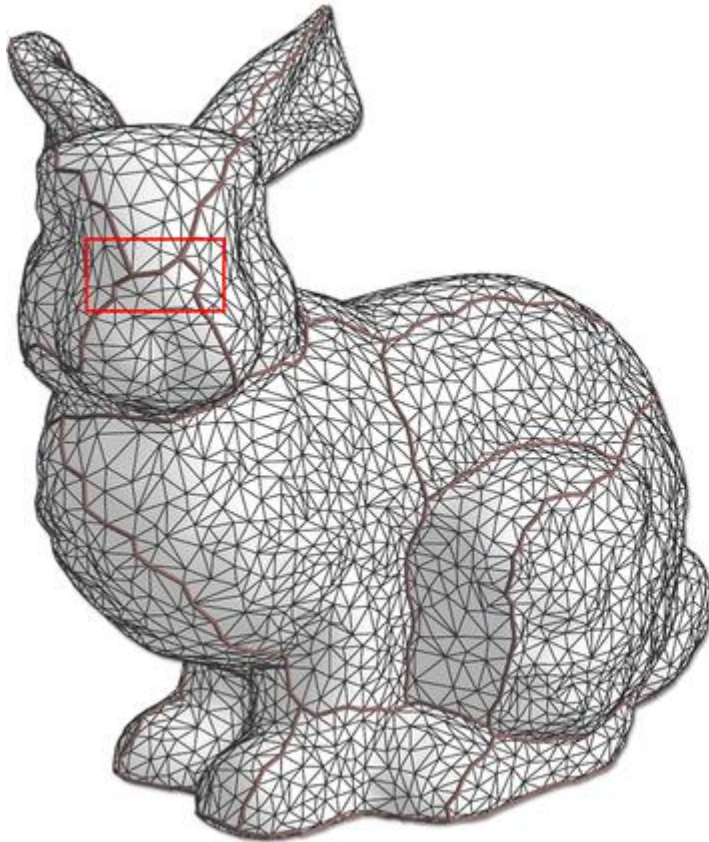
Modeling

- Build 3D representation of the virtual world
- The process of generating “data” in computer graphics

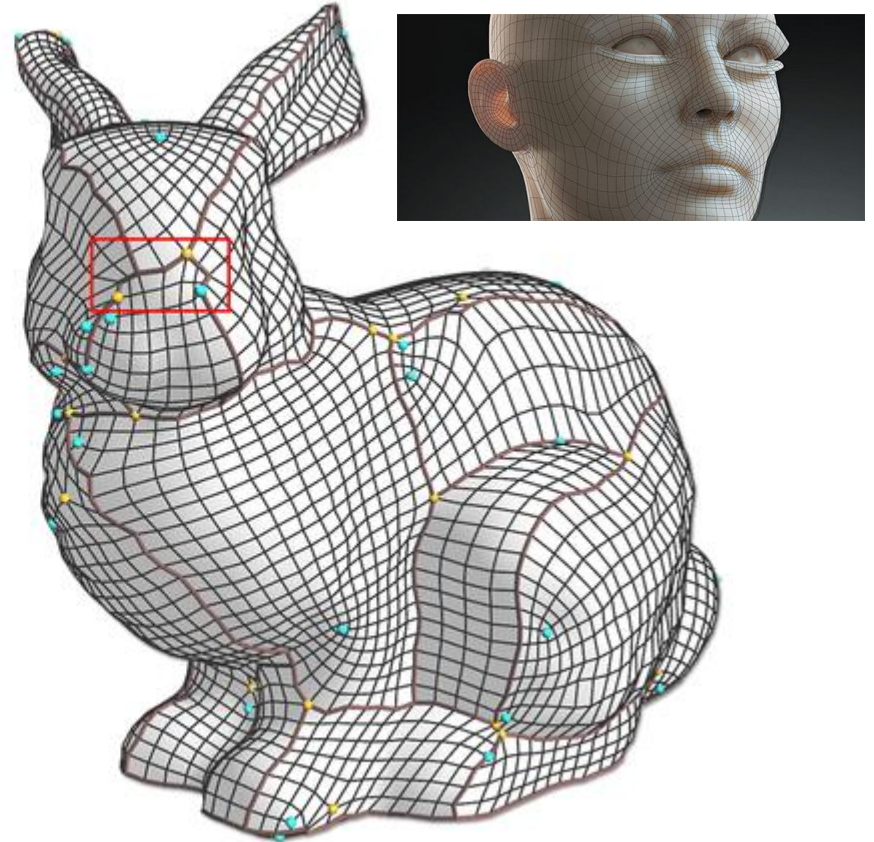


Modeling (cont.)

- Meshes



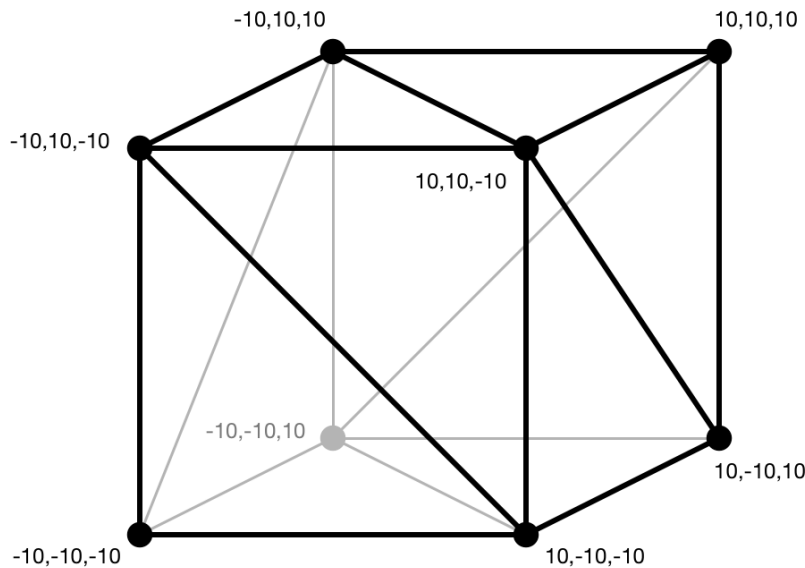
triangle mesh



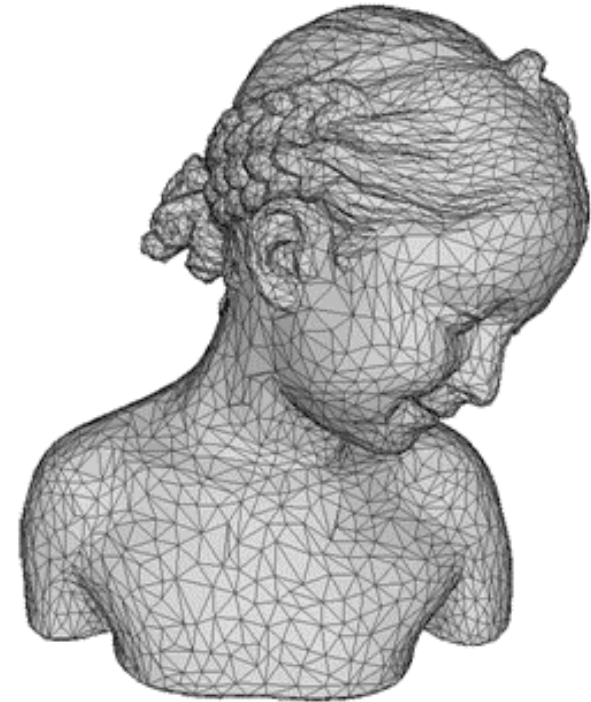
quad mesh

Modeling (cont.)

- **Triangle mesh** is the most popular representation
- Define the **positions** and **adjacencies** of **vertices**



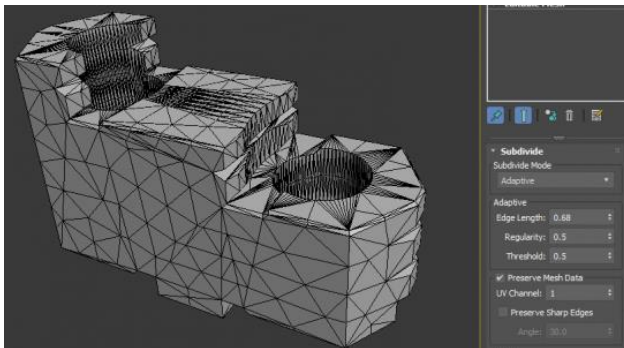
12 triangles



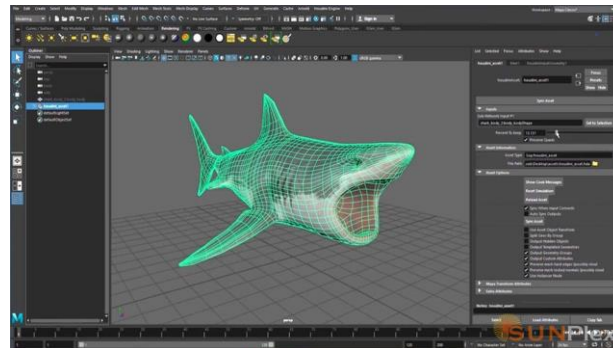
10K triangles

Modeling (cont.)

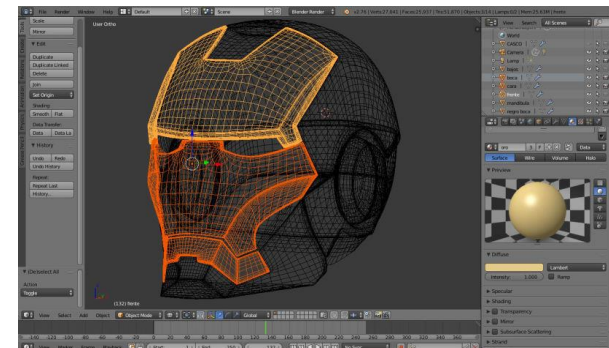
- 3D models are usually obtained by professional manipulations in 3D modeling tools



 Blender



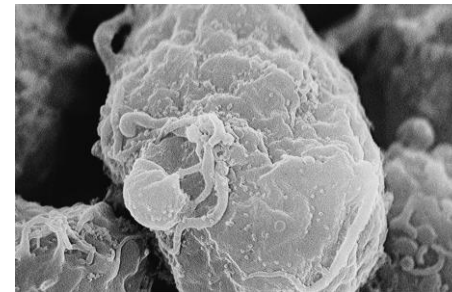
 Maya



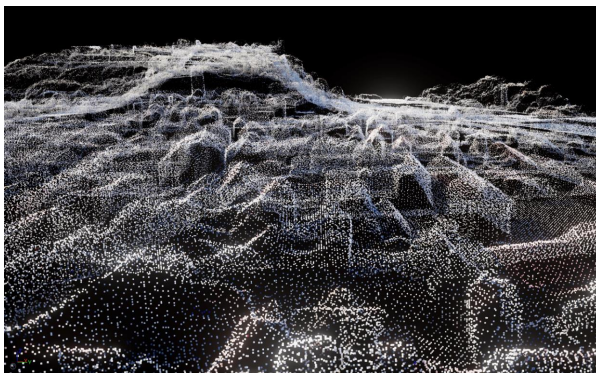
 AUTODESK
3DS MAX

Modeling (cont.)

- World geometries are diverse!



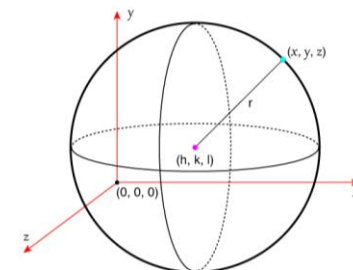
- There are alternative geometry representations



point cloud



volume data



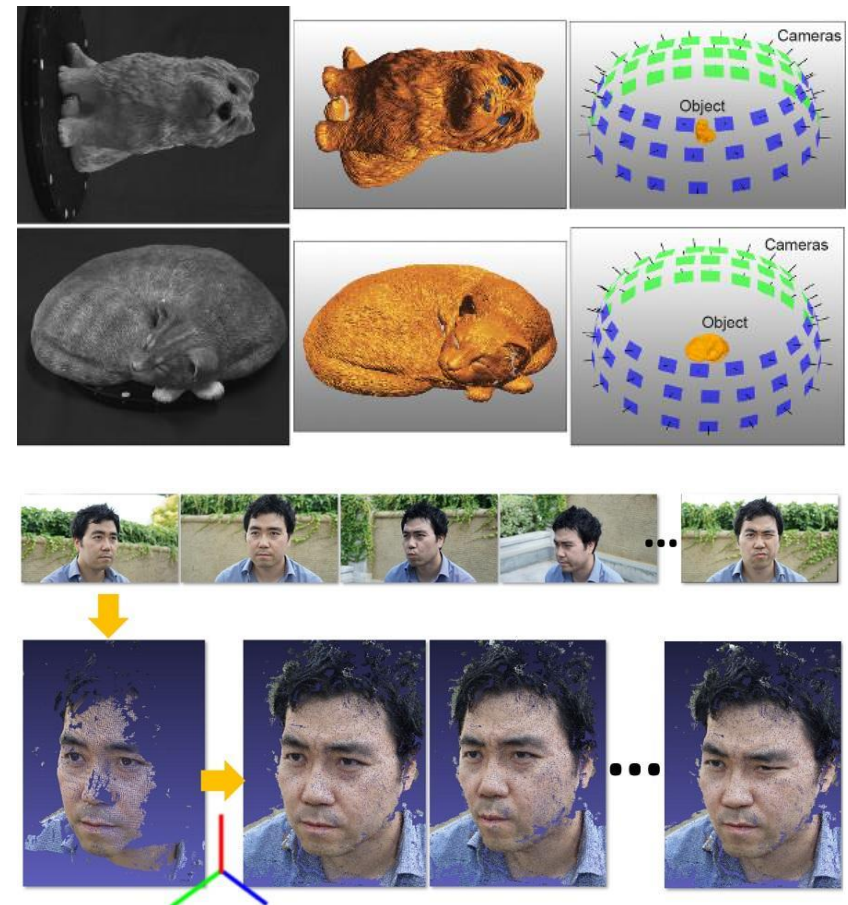
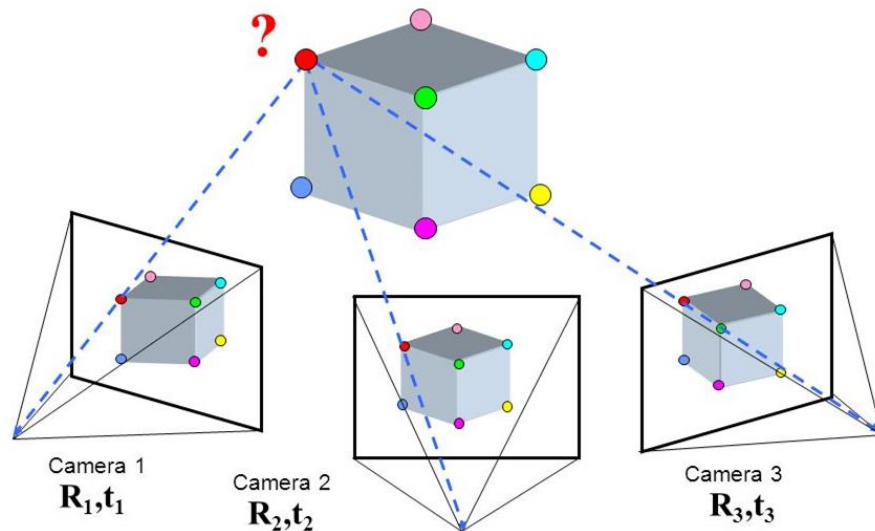
$$(x - h)^2 + (y - k)^2 + (z - l)^2 = r^2$$

here, r = radius, (h, k, l) = center
 (x, y, z) = any point on the surface

functions

Modeling (cont.)

- Multi-view reconstruction



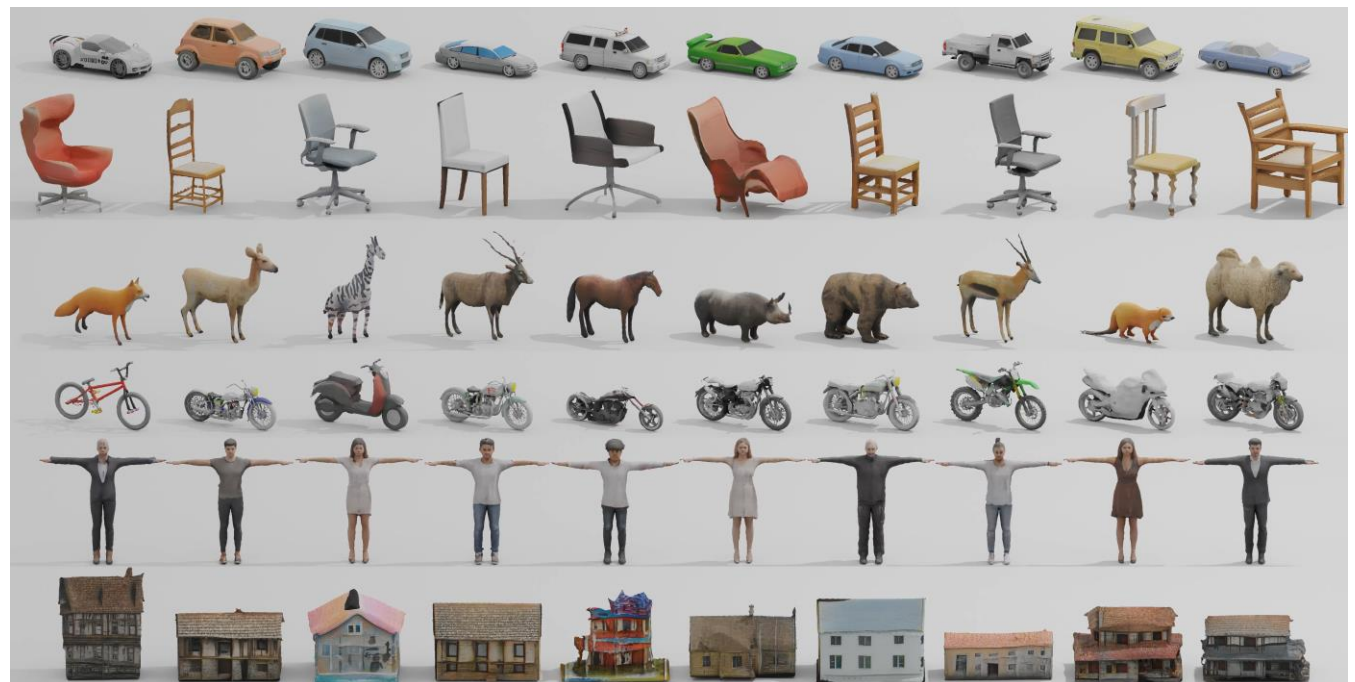
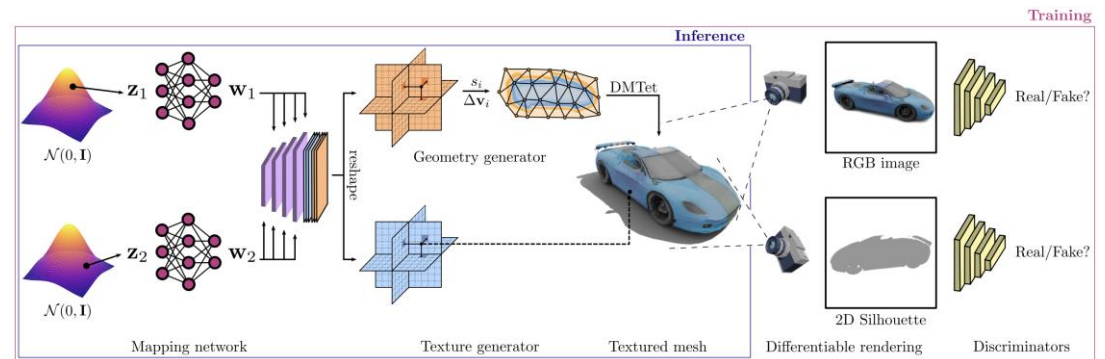
Modeling (cont.)

- 3D scanning



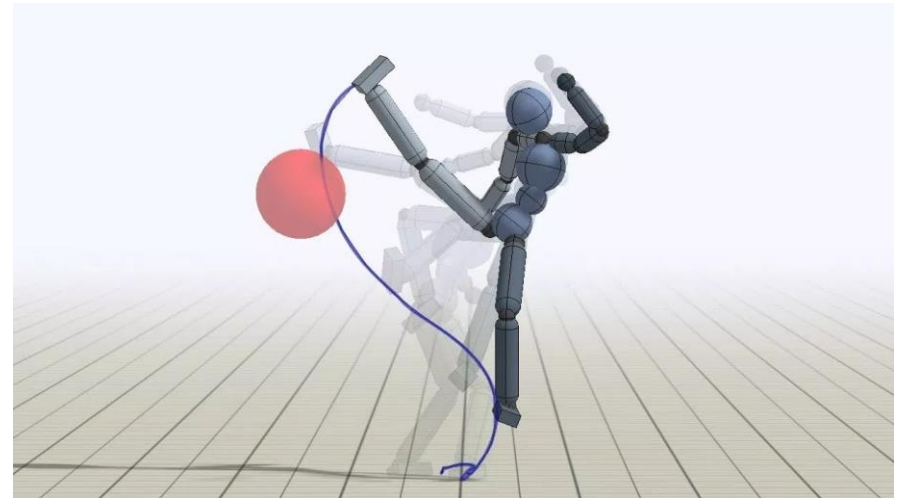
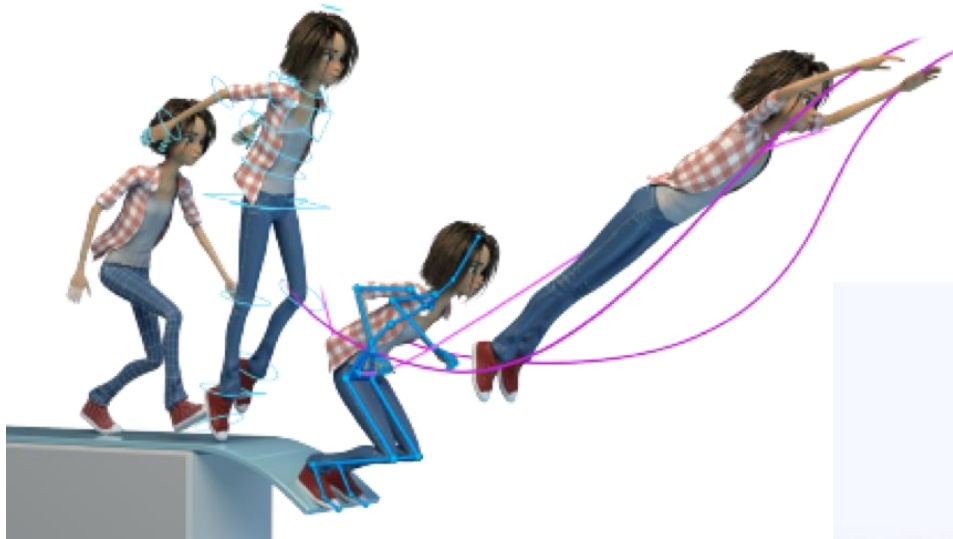
Modeling (cont.)

- AI generated



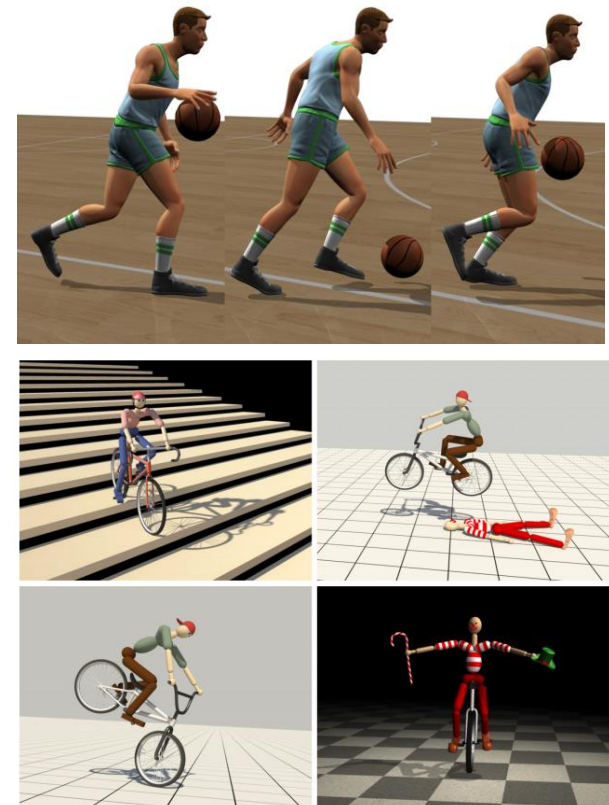
Animation

- Describe (or simulate) how the geometry changes / moves over time



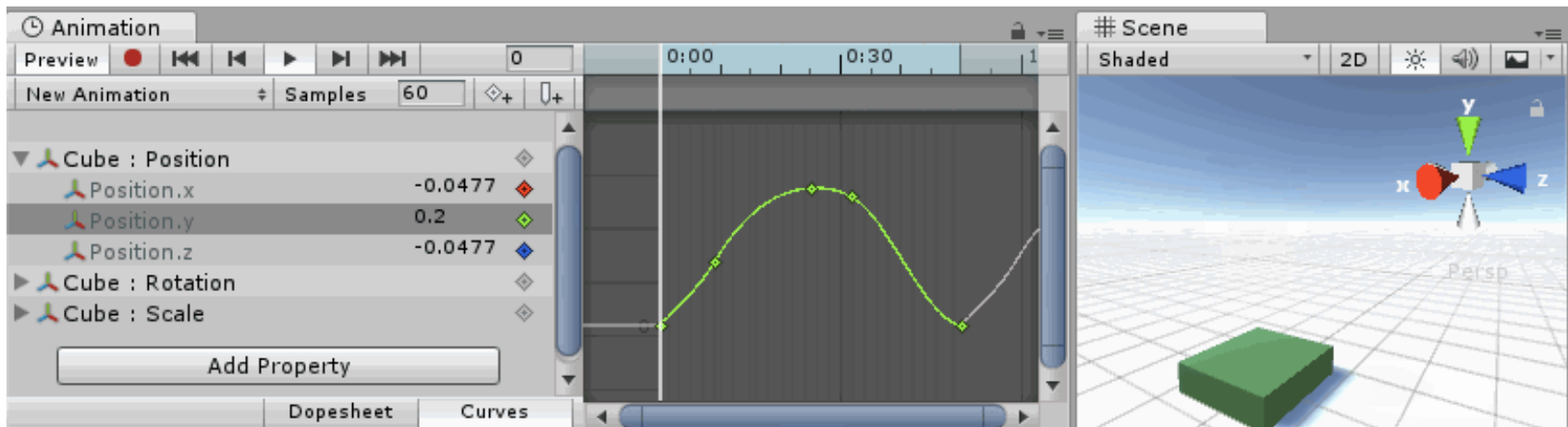
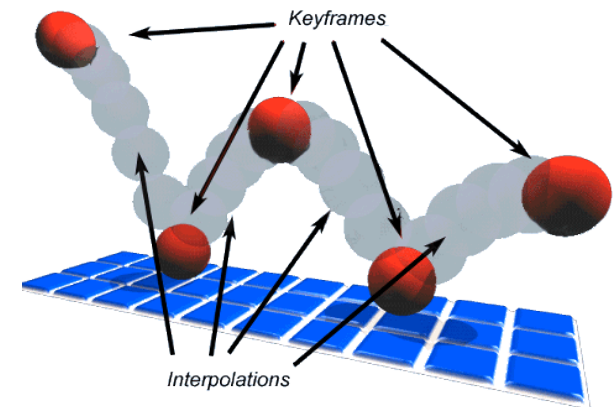
Animation (cont.)

- Animations are usually expected to be physically-based



Animation (cont.)

- Keyframe-based animations

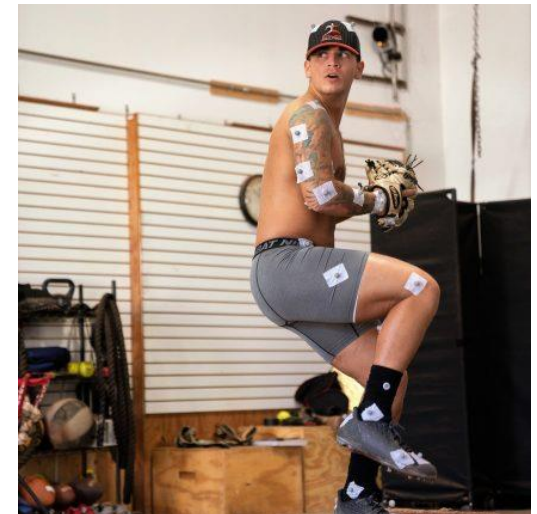


Animation (cont.)

- Motion capture

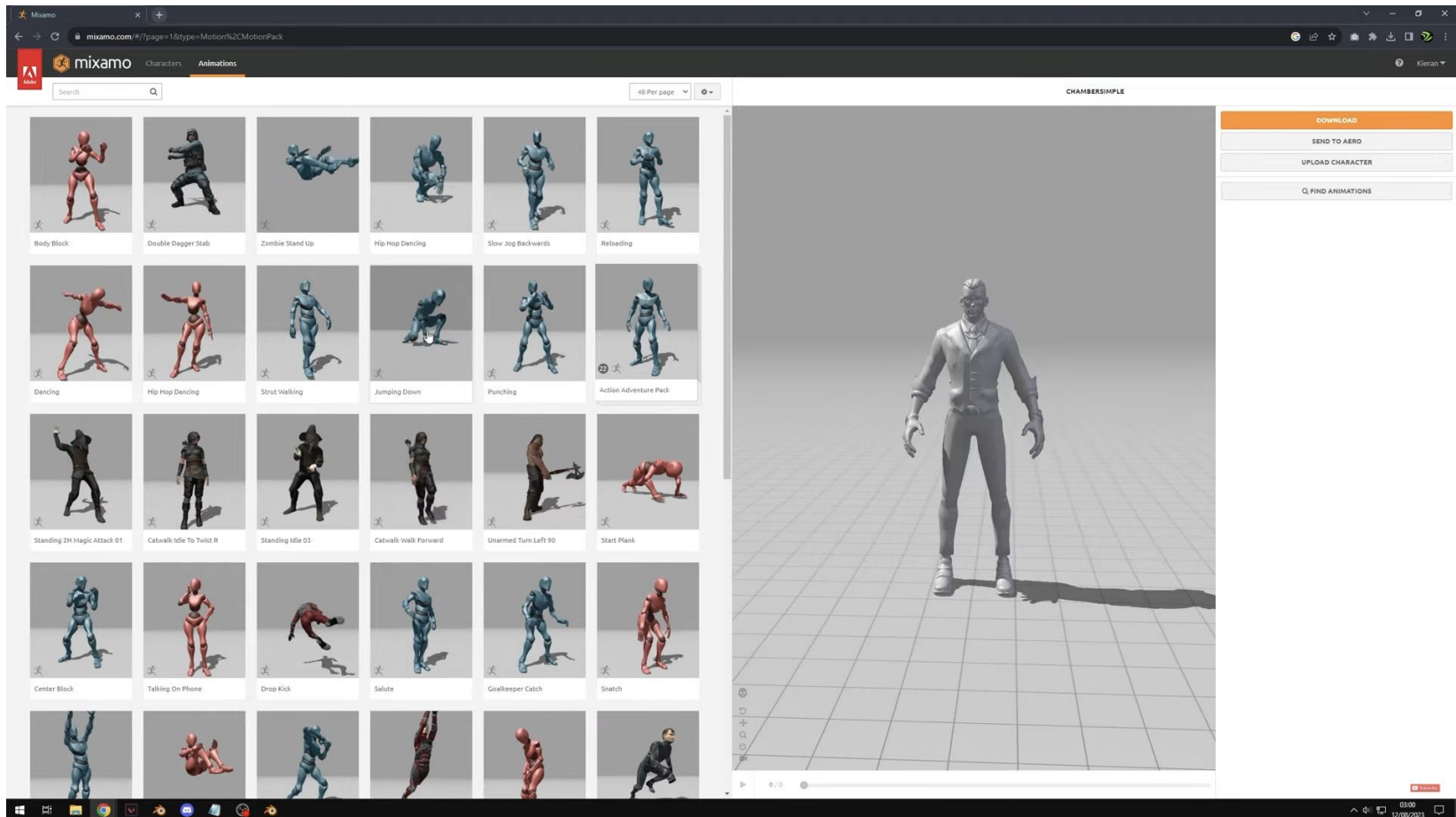


Dawn of the Planet of the Apes, 2014



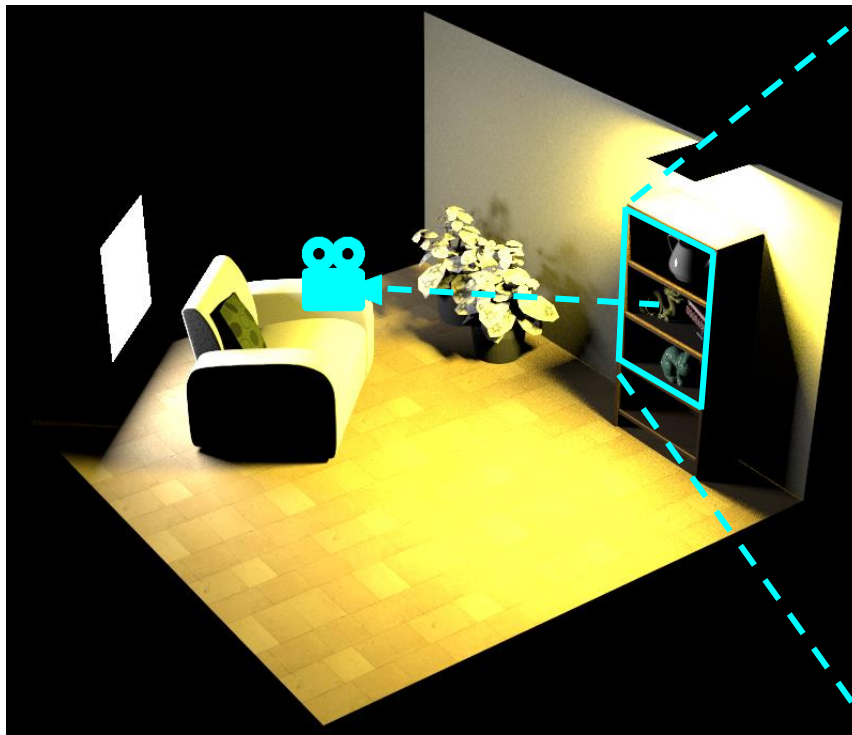
Animation (cont.)

- AI generated



Rendering

- Simulate the appearance of virtual objects and synthesize the final image



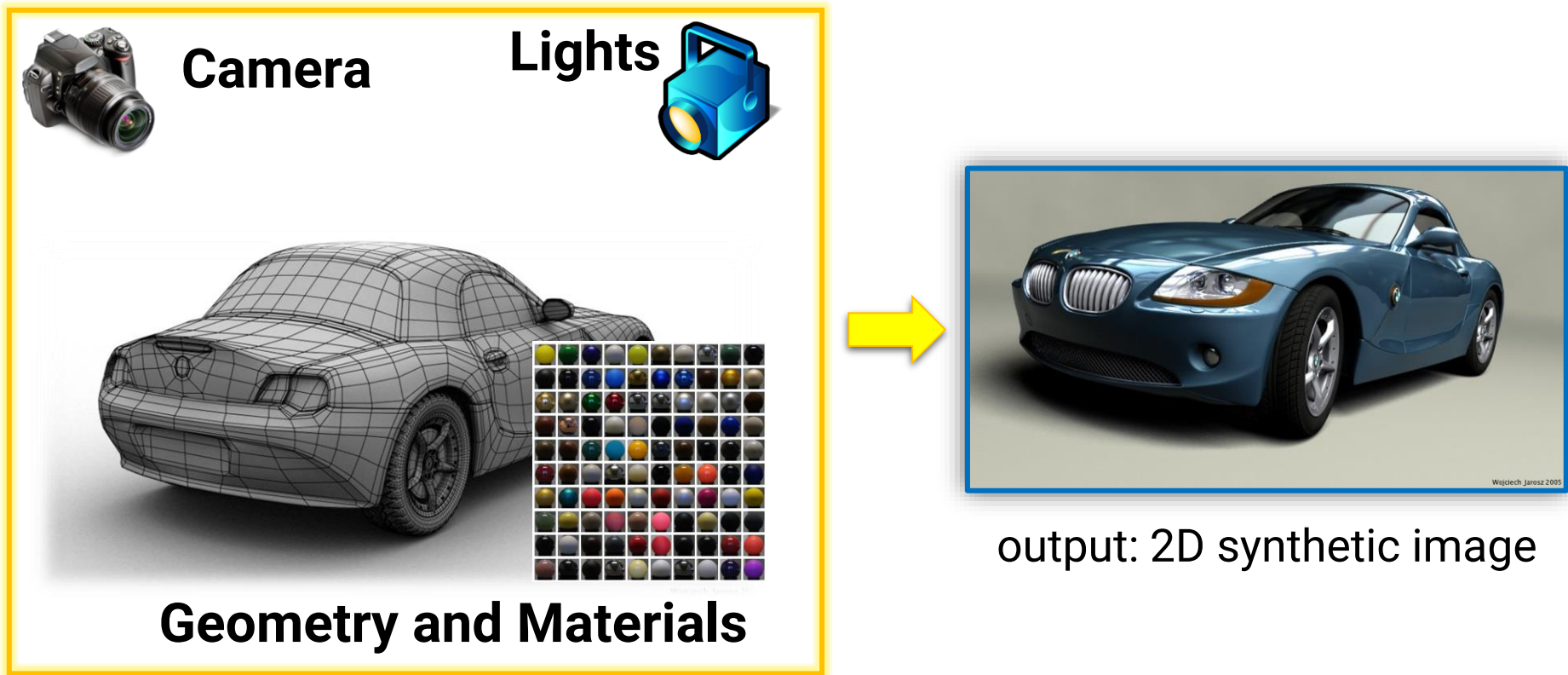
3D virtual world



rendered image

Rendering (cont.)

- Simulate the appearance of virtual objects and synthesize the final image

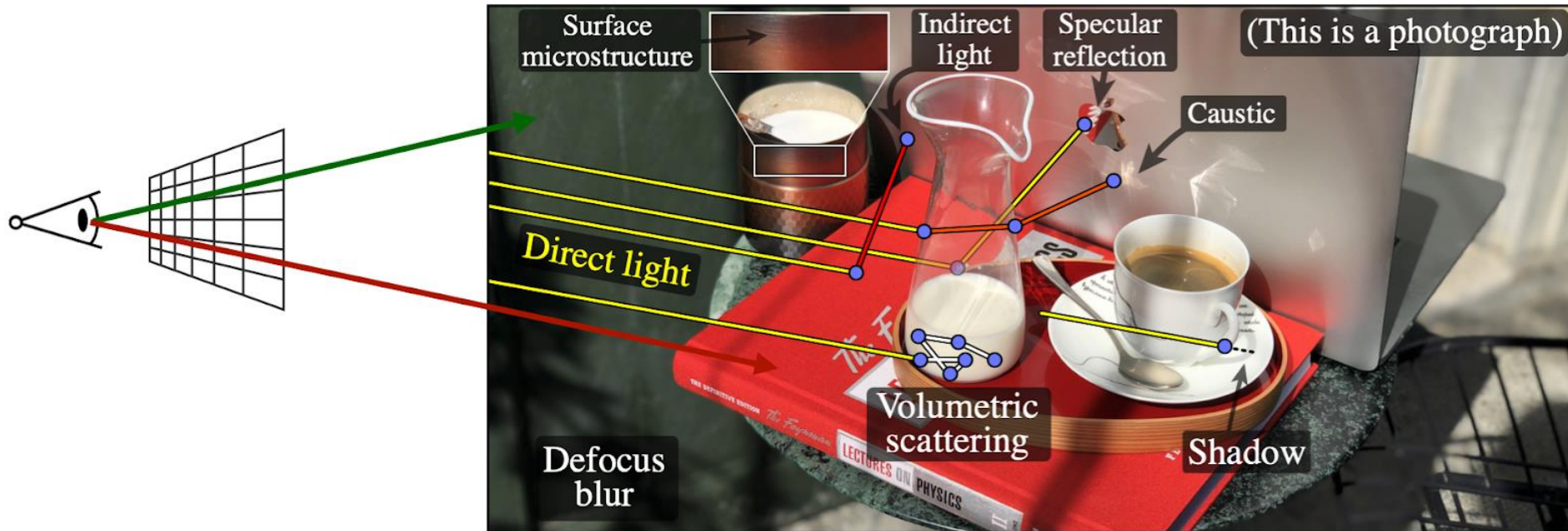


input: 3D description of a scene

Rendering (cont.)

- **Physically-based rendering**

- Uses **physics** and **math** to simulate the interaction between matter and light, **realism** is the primary goal



Rendering (cont.)

- Non-photo-realistic rendering

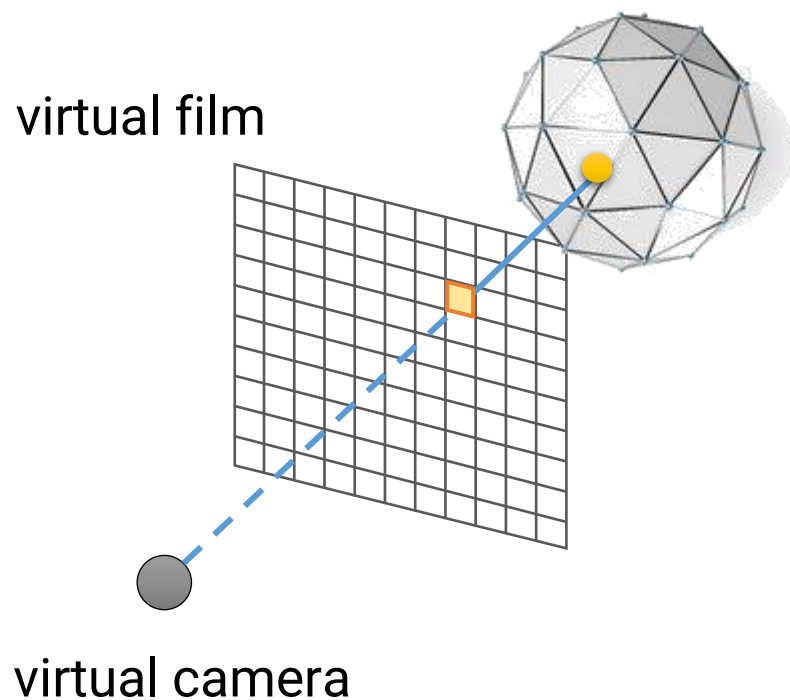


Copyright ©七龍珠 電光炸裂！ZERO, 2024, Bandai Namco Entertainment Inc.

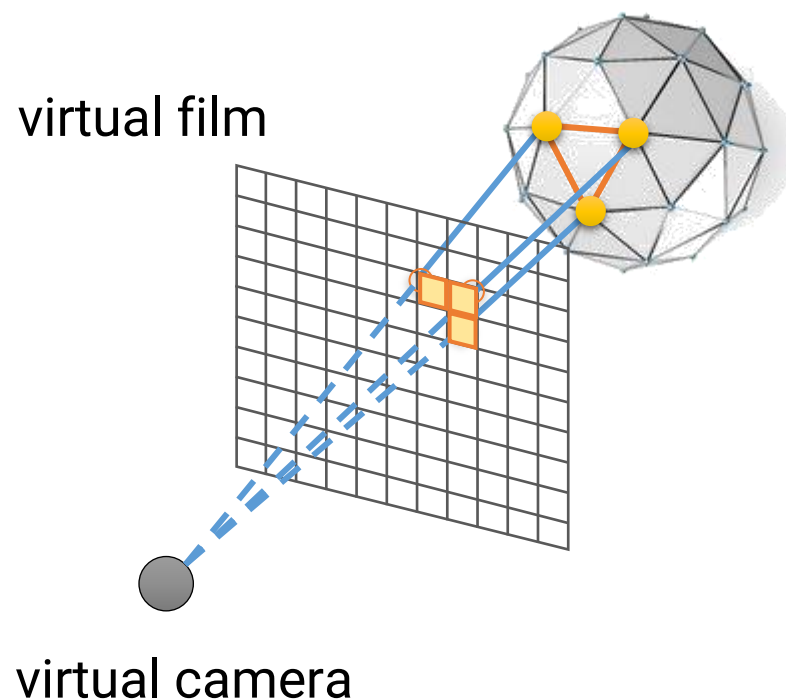
Rendering (cont.)

- Two ways for generating synthetic images

Ray tracing



Rasterization



Rendering (cont.)

- We will focus on the **rasterization-based** rendering because
 - It is widely used in **interactive computer graphics** and has more applications in our daily lives
 - It is more commonly used in Taiwan's industry
 - Thus, can be a great help to your future jobs
 - It takes less time to generate an image
- However, the knowledge is the same and we will also give an overview of ray tracing at the end of this course

Case Study: Animation Production Pipeline

Animation Production Pipeline



story



text treatment



storyboard



voice

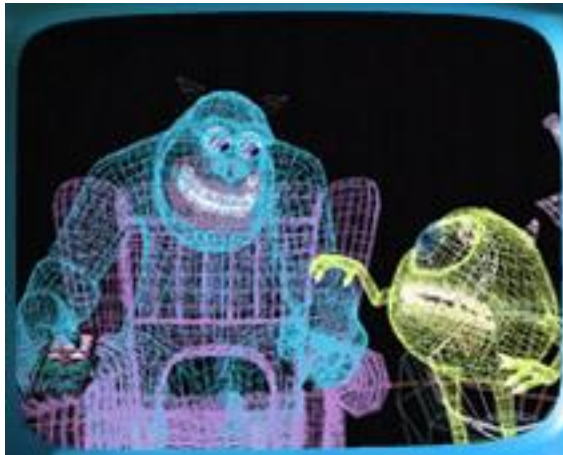


storyreel



look and feel

Animation Production Pipeline (cont.)



modeling / articulation



layout



animation



shading / lighting



rendering



final touch

Outline

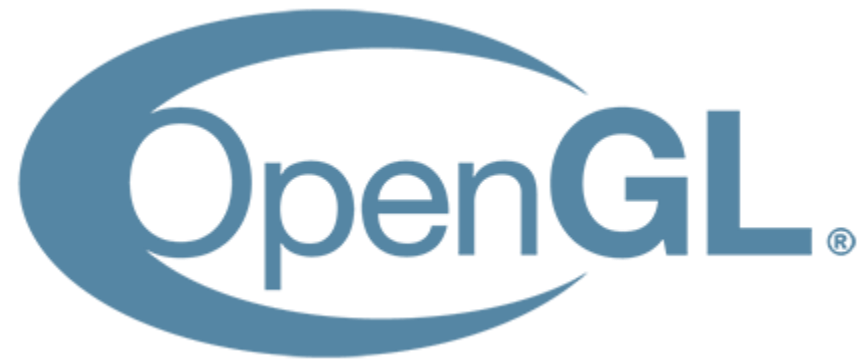
- Introduction to computer graphics
- **Introduction to graphics programming**
- Homework assignments and rendering competition

Graphics Programming

- For rasterization-based graphics, programs are usually implemented with graphics **application programming interface (API)** and **shader programs**
- Common choices are
 - OpenGL + GLSL (OpenGL shading language)
 - OpenGL ES
 - WebGL
 - DirectX + HLSL (High-level shading language)
 - Vulkan + GLSL/HLSL

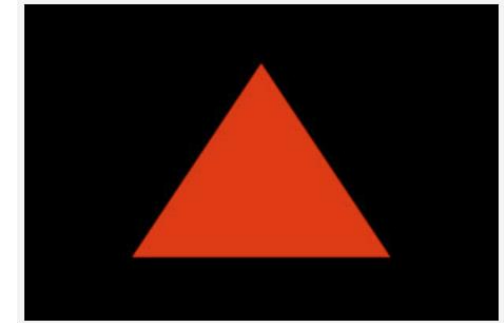
OpenGL

- A **cross-platform** API for rendering 2D and 3D vector graphics, typically used to interact with a graphics processing unit (GPU)
- Developed by Silicon Graphics Inc. (SGI) in 1991
- Managed by a non-profit technology consortium **Khronos Group** after 2006



OpenGL + GLSL

- A simple program to draw a triangle on the screen
 - 176 lines of C++ code and 16 lines of shader code



```

32 static void RenderSceneCB()
33 {
34     glClearColor(GL_COLOR_BUFFER_BIT);
35
36     glBindBuffer(GL_ARRAY_BUFFER, VBO);
37
38     glEnableVertexAttribArray(0);
39
40     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
41
42     glDrawArrays(GL_TRIANGLES, 0, 3);
43
44     glDisableVertexAttribArray(0);
45
46     glutSwapBuffers();
47 }
48
49
50 static void CreateVertexBuffer()
51 {
52     Vector3f Vertices[3];
53     Vertices[0] = Vector3f(-1.0f, -1.0f, 0.0f); // bottom left
54     Vertices[1] = Vector3f(1.0f, -1.0f, 0.0f); // bottom right
55     Vertices[2] = Vector3f(0.0f, 1.0f, 0.0f); // top
56
57     glGenBuffers(1, &VBO);
58     glBindBuffer(GL_ARRAY_BUFFER, VBO);
59     glBufferData(GL_ARRAY_BUFFER, sizeof(Vector3f) * 3, Vertices, GL_STATIC_DRAW);
60 }

```

```

#version 330 core

layout (location = 0) in vec3 Position;

void main()
{
    gl_Position = vec4(0.5 * Position.x, 0.5 * Position.y, Position.z, 1.0);
}

#version 330 core

out vec4 FragColor;

void main()
{
    FragColor = vec4(1.0, 0.0, 0.0, 0.0);
}

```

Why not Teaching Vulkan in this Course?

- A simple program to draw a triangle on the screen
 - **457** lines of C++ code

```
void CreateSwapChain();
void CreateCommandBuffer();
void CreateRenderPass();
void CreateFramebuffer();
void CreateShaders();
void CreatePipeline();
void RecordCommandBuffers();
void RenderScene();

std::string m_appName;
VulkanWindowControl* m_pWindowControl;
OgldevVulkanCore m_core;
std::vector<VkImage> m_images;
VkSwapchainKHR m_swapChainKHR;
VkQueue m_queue;
std::vector<VkCommandBuffer> m_cmdBufs;
VkCommandPool m_cmdBufPool;
std::vector<VkImageView> m_views;
VkRenderPass m_renderPass;
std::vector<VkFramebuffer> m_fbs;
VkShaderModule m_vsModule;
VkShaderModule m_fsModule;
VkPipeline m_pipeline;
};
```

...

```
rastCreateInfo.polygonMode = VK_POLYGON_MODE_FILL;
rastCreateInfo.cullMode = VK_CULL_MODE_BACK_BIT;
rastCreateInfo.frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
rastCreateInfo.lineWidth = 1.0f;

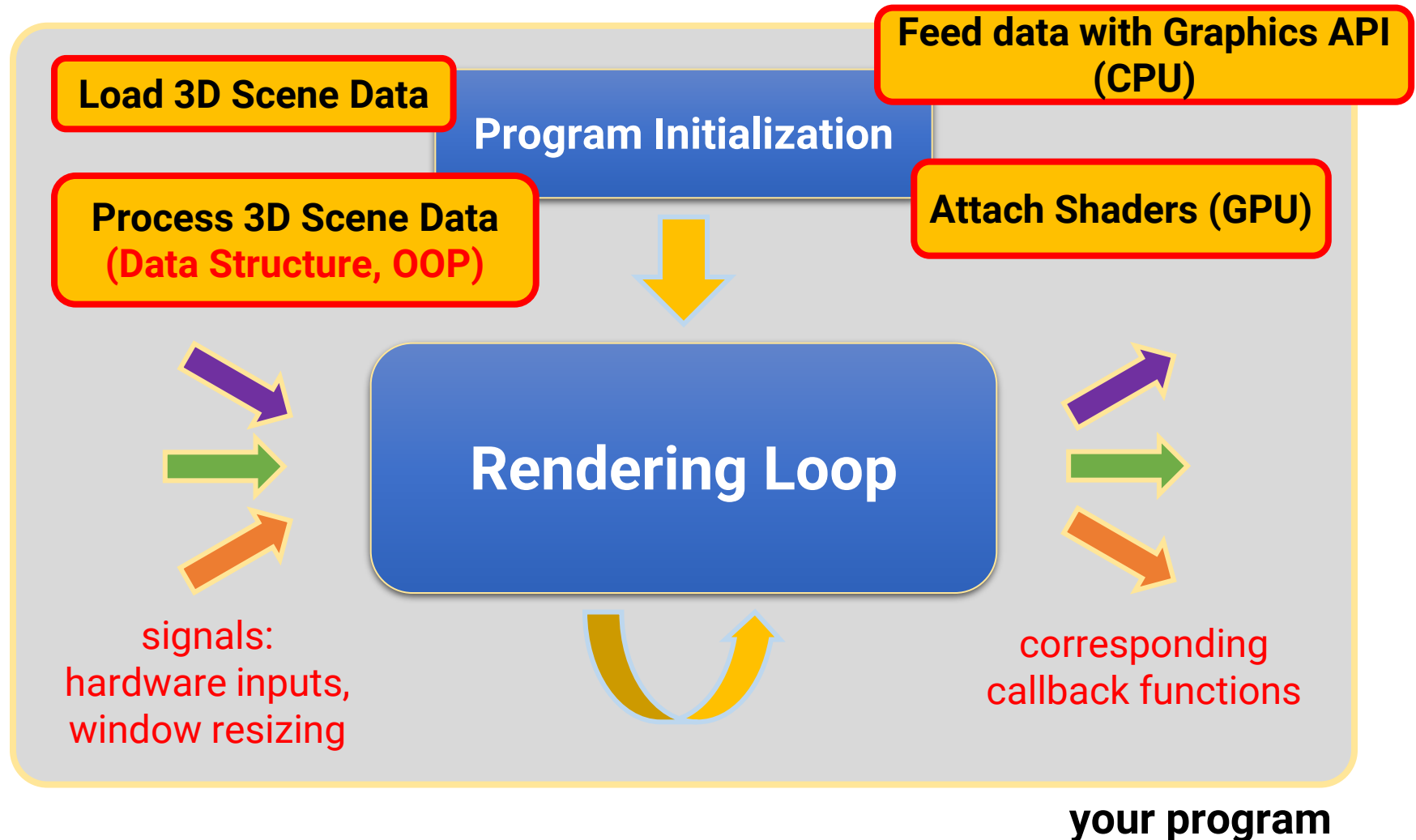
VkPipelineMultisampleStateCreateInfo pipelineMSSCreateInfo = {};
pipelineMSSCreateInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_

VkPipelineColorBlendAttachmentState blendAttachState = {};
blendAttachState.colorWriteMask = 0xf;

VkPipelineColorBlendStateCreateInfo blendCreateInfo = {};
blendCreateInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREAT
blendCreateInfo.logicOp = VK_LOGIC_OP_COPY;
blendCreateInfo.attachmentCount = 1;
blendCreateInfo.pAttachments = &blendAttachState;

VkGraphicsPipelineCreateInfo pipelineInfo = {};
pipelineInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
pipelineInfo.stageCount = ARRAY_SIZE_IN_ELEMENTS(shaderStageCreateInfo);
pipelineInfo.pStages = &shaderStageCreateInfo[0];
pipelineInfo.pVertexInputState = &vertexInputInfo;
pipelineInfo.pInputAssemblyState = &pipelineIACreateInfo;
pipelineInfo.pViewportState = &vpCreateInfo;
pipelineInfo.pRasterizationState = &rastCreateInfo;
```

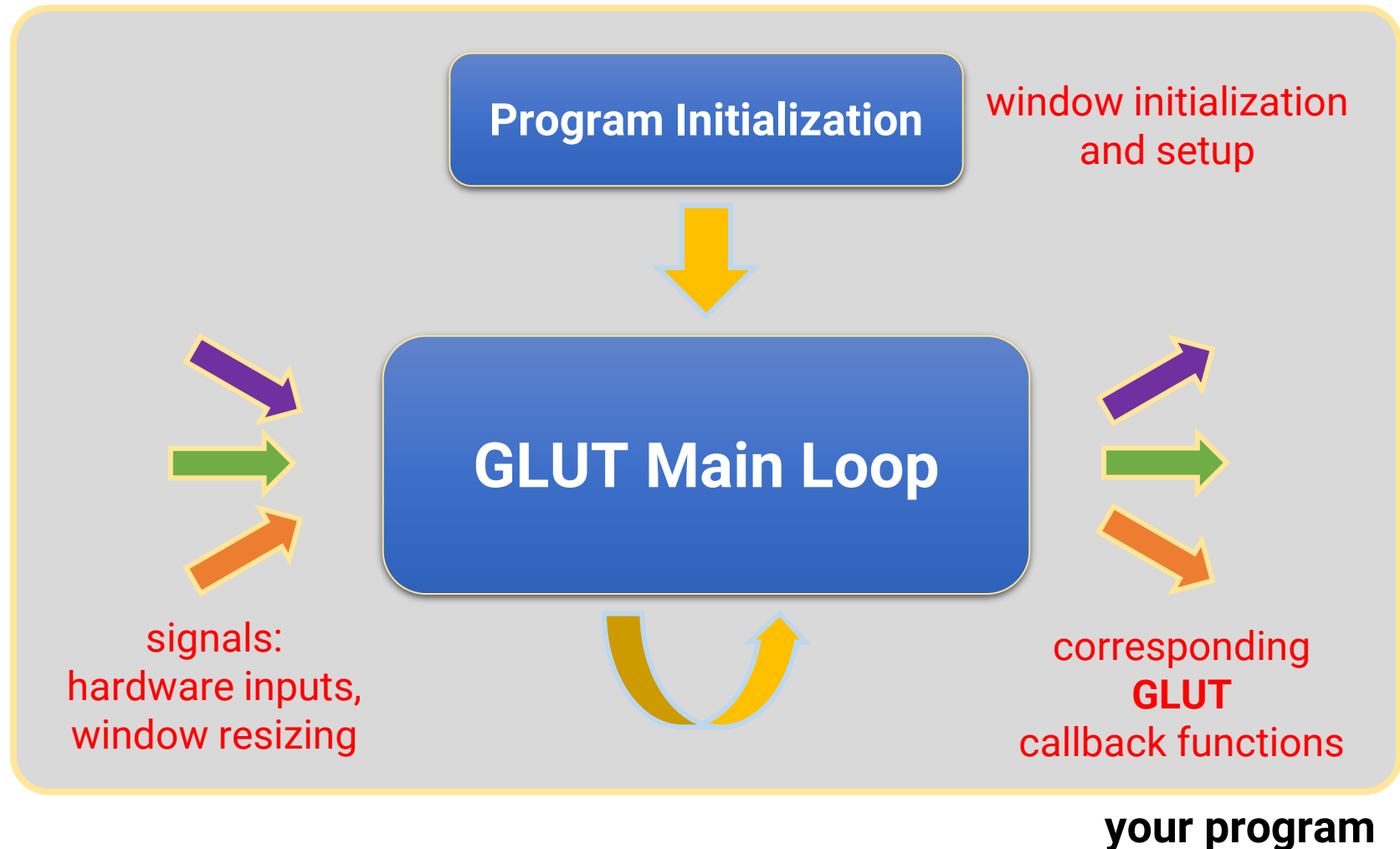
Life Cycle of a Rendering Engine



Library for Handling Screen Rendering

- **GLUT: OpenGL Utility Toolkit ([link](#))**
 - Window system independent
 - Implement a simple window application programming interface (API) for OpenGL
 - Designed for constructing small to medium-sized OpenGL programs
 - For large applications, it is suggested to use a native window system toolkit such as Qt for a more sophisticated UI
- **FreeGLUT: Free OpenGL Utility Toolkit ([link](#))**
 - GLUT has gone into stagnation and has some issues with licenses
 - FreeGLUT is intended to be a full replacement for GLUT

Life Cycle of a FreeGLUT Program



Structure of a FreeGLUT Program

```
// OpenGL and FreeGlut headers.
```

```
#include <freeglut.h>
```

```
int main(int argc, char** argv)
```

```
{
```

```
    // Setting window properties.
```

```
    glutInit(&argc, argv);
```

```
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
```

```
    glutInitWindowSize(640, 360);
```

```
    glutInitWindowPosition(100, 100);
```

```
    glutCreateWindow("OpenGL Renderer");
```

create the window
and set window
properties

```
    // Initialization.
```

```
    SetupRenderState();
```

do initialization
jobs

```
    // Register callback functions.
```

```
    glutDisplayFunc(RenderSceneCB);
```

```
    glutIdleFunc(RenderSceneCB);
```

```
    glutReshapeFunc(ReshapeCB);
```

```
    glutSpecialFunc(ProcessSpecialKeysCB);
```

```
    glutKeyboardFunc(ProcessKeysCB);
```

register callback
functions

```
    // Start rendering loop.
```

```
    glutMainLoop();
```

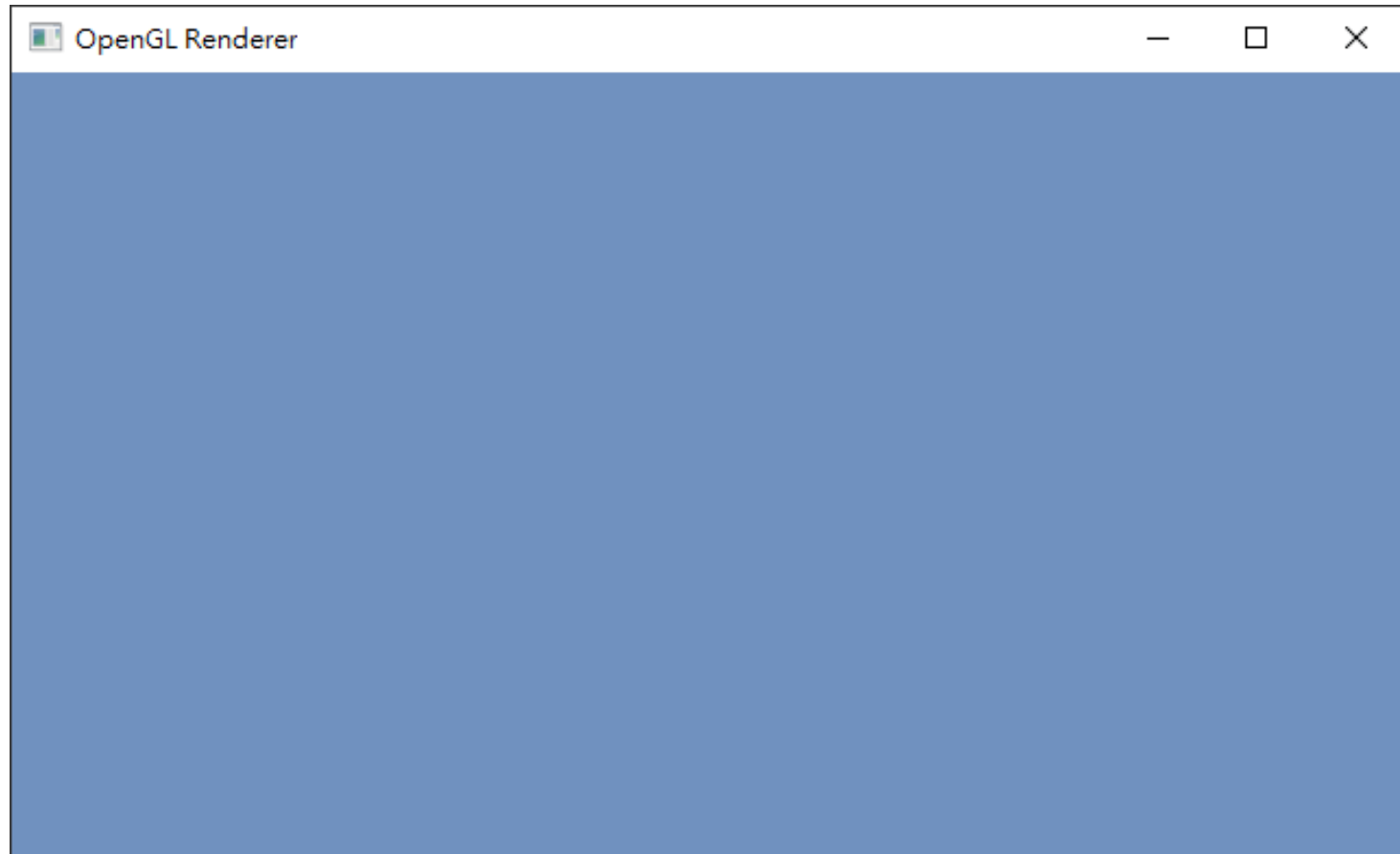
start the
main loop

```
    return 0;
```

```
}
```

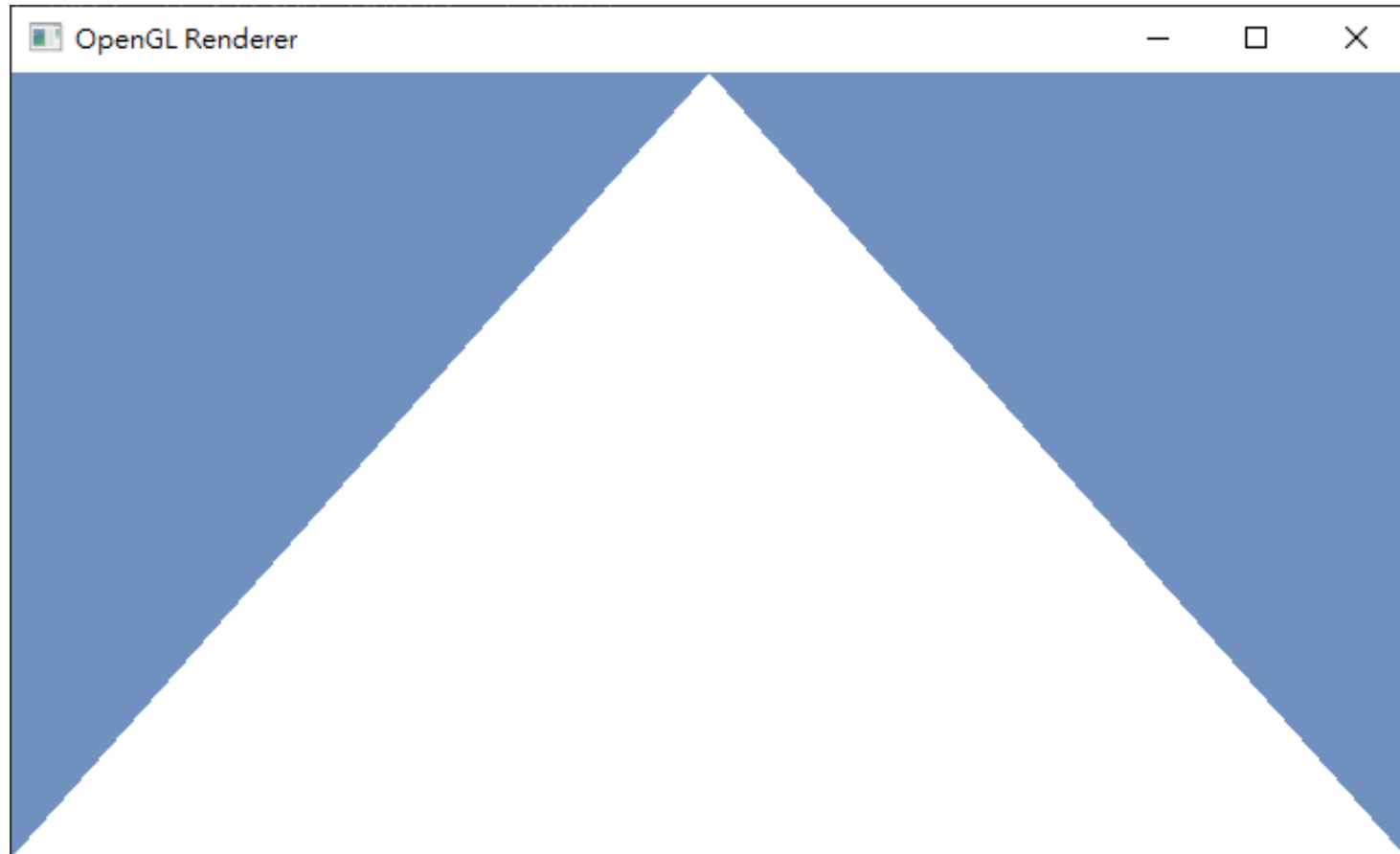
FreeGLUT Window

- FreeGLUT will create and maintain a window on screen



Next Two Weeks

- We will learn how to render a single triangle



Outline

- Introduction to computer graphics
- Introduction to graphics programming
- **Homework assignments and rendering competition**

Topics We Plan to Cover

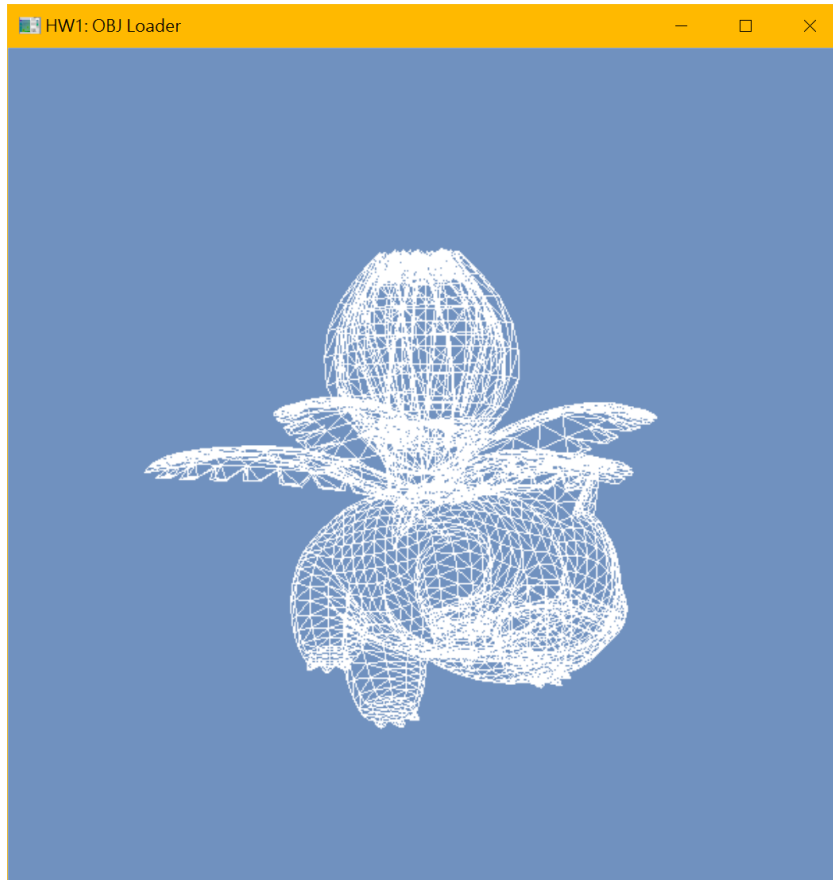
Basic

- HW1** • Geometry Representation
 - Transformations
 - Camera
- HW2** • GPU Graphics Pipeline
 - Shading
- HW3** • Textures

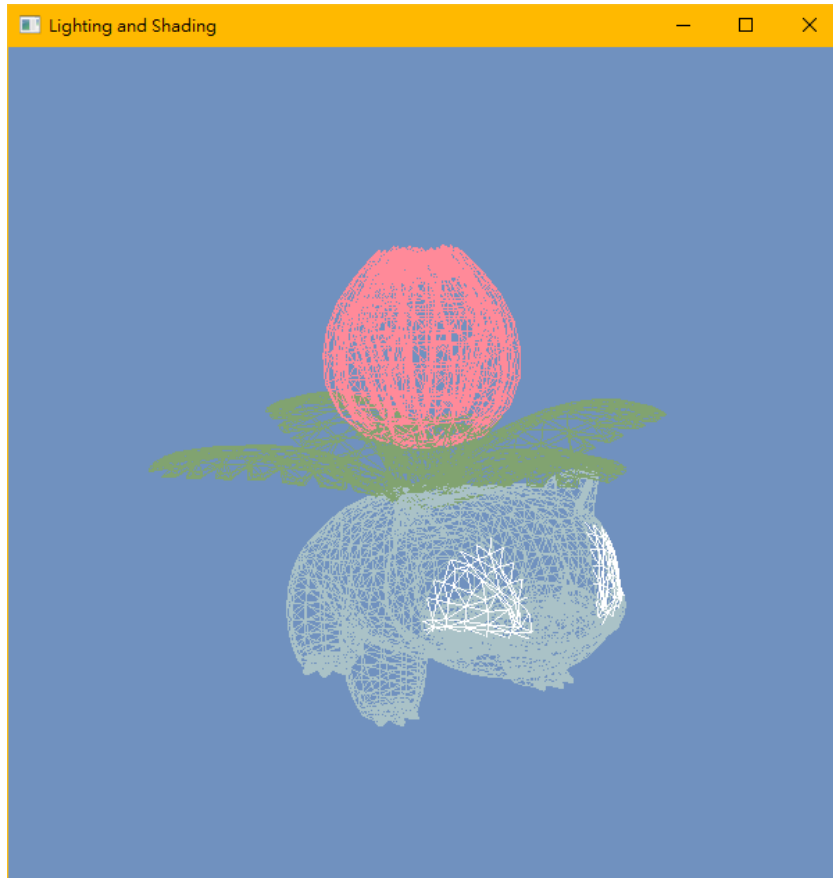
Advanced

- Transparency
- Shadows
- Deferred Shading
- Terrain
- Ray Tracing
- Advanced Shaders
- Unity Case Study

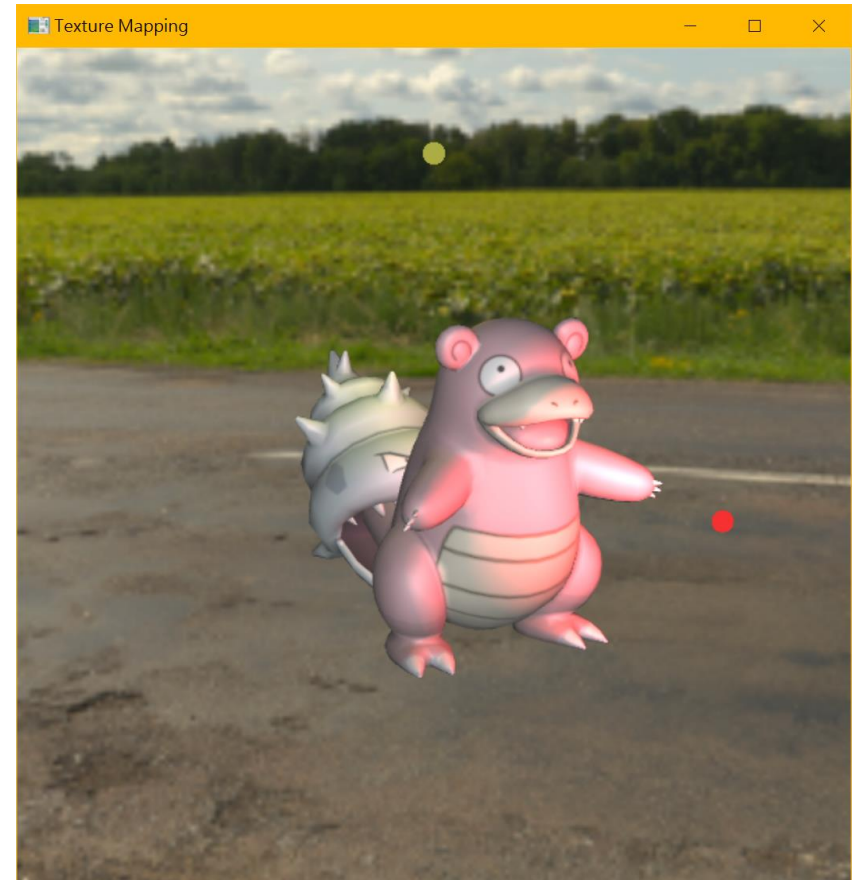
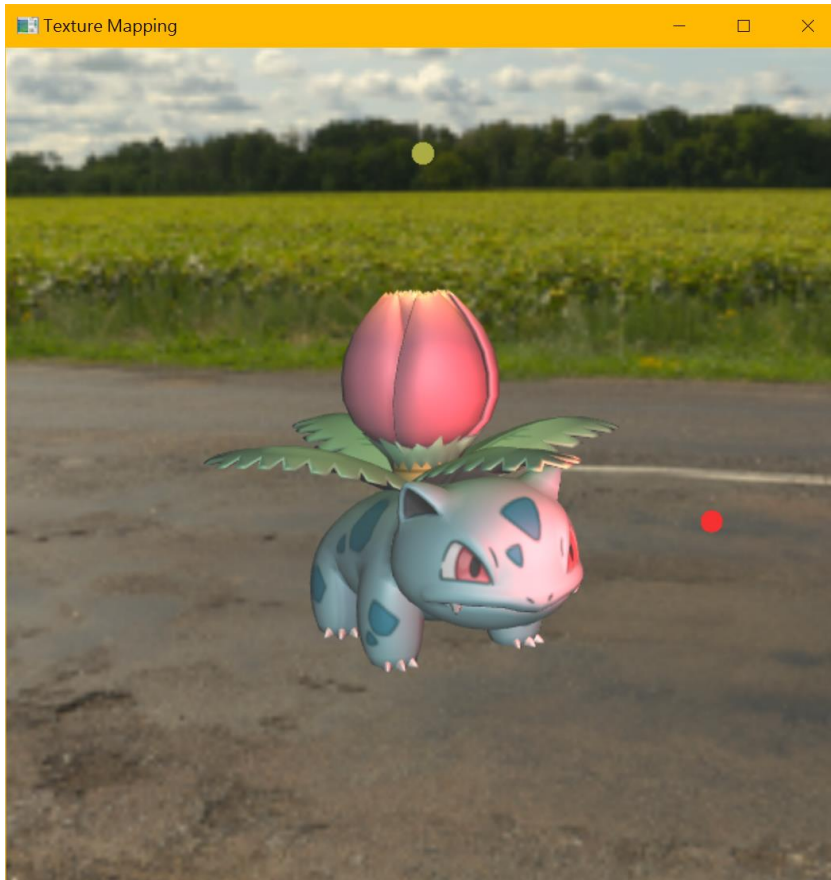
HW1: Geometry Representation (15%)



HW2: GPU Pipeline and Materials (15%)



HW3: Lighting and Texturing (15%)



Rendering Competition (5%)

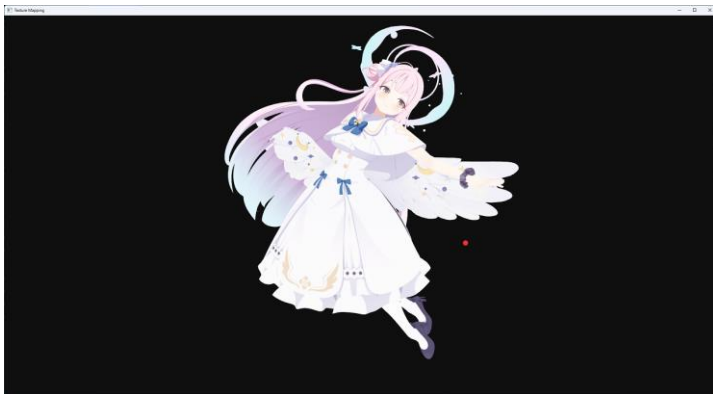
- **Submit a beautiful image rendered by your program**
- Your program is encouraged to support the following features
 - Advanced rendering algorithms
 - Multiple objects
 - New 3D models downloaded from the Internet
 - New skybox downloaded from the Internet
 - Nice lighting and material setting
 - ... etc.

Rendering Competition (5%)



Rendering Competition (5%)

- Amazing works from previous courses



彭東駿



陳映璋



張智堯

