Course Overview

Computer Graphics

Yu-Ting Wu

Outline

- Course information, policy, and rules
- Introduction to computer graphics
- Introduction to graphics programming
- Homework assignments and rendering competition

Outline

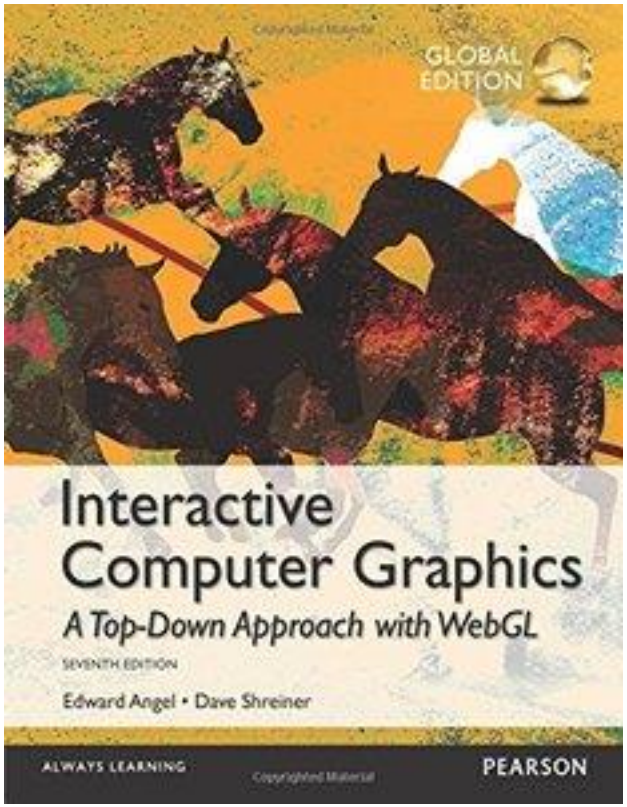
- **Course information, policy, and rules**
- Introduction to computer graphics
- Introduction to graphics programming
- Homework assignments and rendering competition

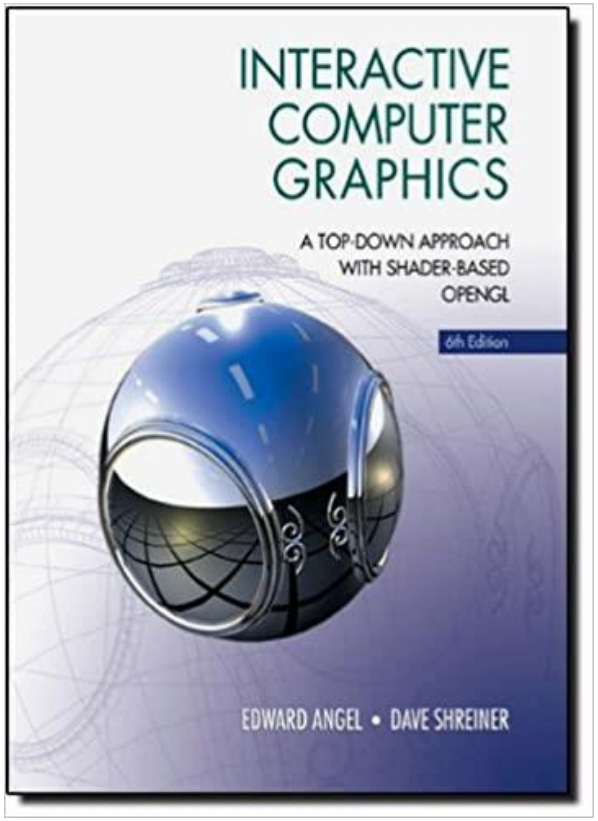
Course Information

- **Meeting time:** 09:10 - 12:00, Monday
- **Classroom:** 電1F-03
- **Instructor:** 吳昱霆 (Yu-Ting Wu)
- **Teaching assistants:** 曾念馨
- **Course webpage:**
  - *https://kevincosner.github.io/courses/CG2023/*
- **Grading:**
  - Assignments: 45% (3 HWs, 18%+18%+9%)
  - Midterm 25%
  - Final exam: 25%
  - Rendering competition: 5%

Textbook (Optional)

- **Interactive Computer Graphics: A Top-Down Approach with WebGL (7th) / Shader-based OpenGL (6th)**





HW Late Policy HW

- One day 90%
- Two days 80%
- Three days 70%
- Four days 60%
- Five days+ 50%
- E.g., assume the deadline for the HW is 12/24 23:59 and you submit your HW on 12/25, you will get a 10% penalty

- You are encouraged to discuss HWs with your classmates; however, the code should **NOT** be highly similar
  - **If caught, you will get ZERO**

Class Rules

- You are welcome to ask questions
  - Raise your hands anytime in class
  - Send an email to me anytime out of class
  - **Please be polite and always reply to the mail!**

- DO **NOT CHAT** in the class


!
CAUTION

We are Going to Write Lots of Codes

The composition of this course:

- Learn the basic concepts of **3D** computer graphics, especially in **modeling** and **rendering**

50%

- Learn how to program with **graphics API (OpenGL)**

50%

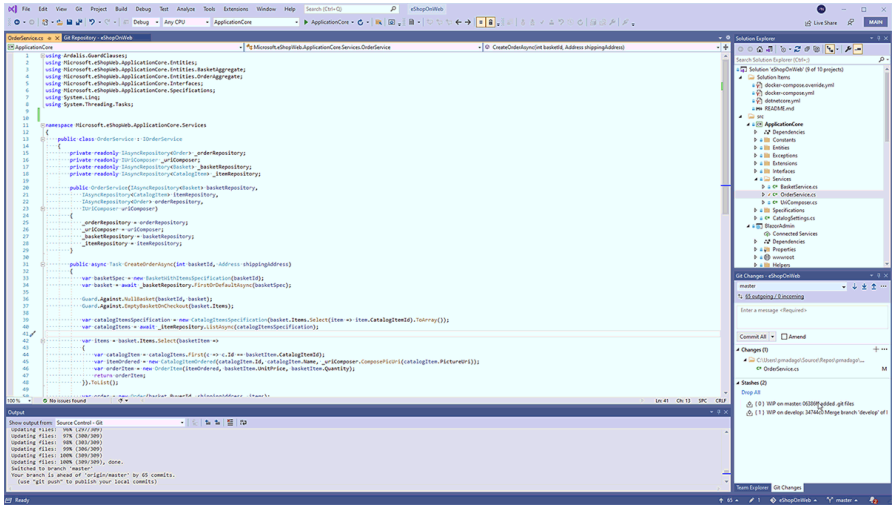
Prerequisites

- **C++ programming experience is required**
- Basic knowledge of **data structure** and **objected-oriented programming** is essential

- It is a **plus** if you
  - Are familiar with **linear algebra**
  - Have taken my course, **multimedia technology and applications**
  - Have experience in **image processing**

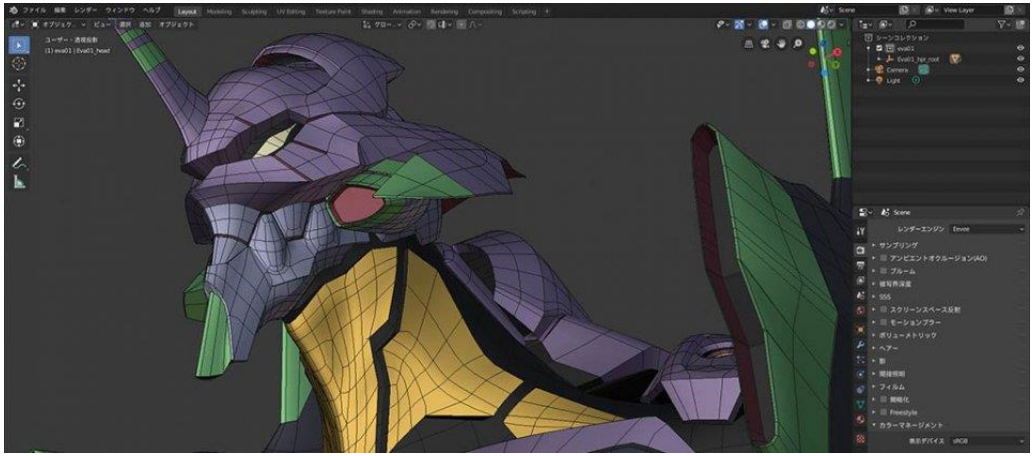
Prerequisites (cont.)

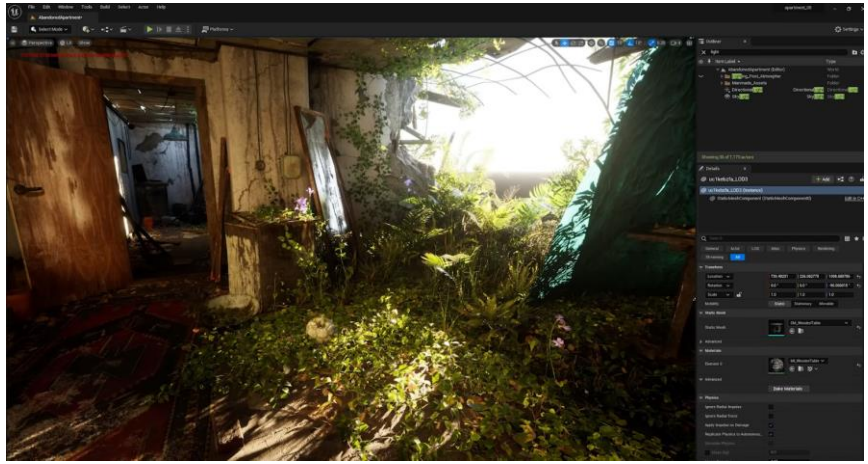
- For all homework assignments, we will provide a skeleton code of the **Visual Studio Community 2022 Project** on **Windows**
  - Download the free IDE from https://visualstudio.microsoft.com/zh-hant/vs/community/



This course is NOT about using Editors

- Instead, we learn the techniques behind the software!





Outline

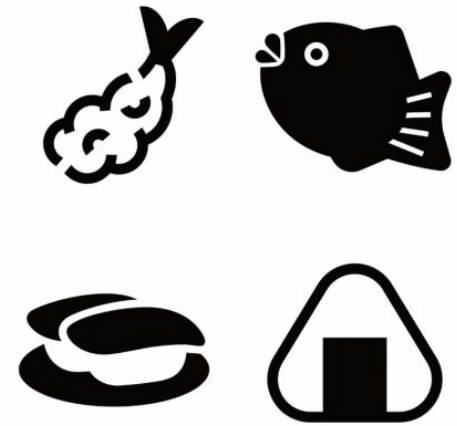
- Course information, policy, and rules
- **Introduction to computer graphics**
- Introduction to graphics programming
- Homework assignments and rendering competition

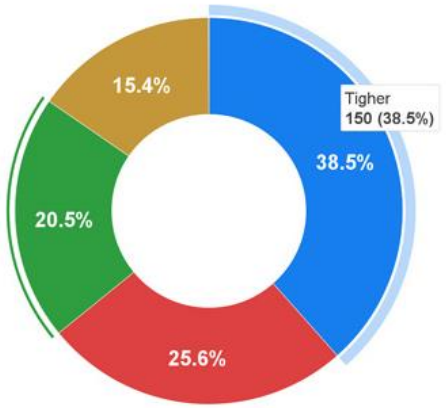
Overview

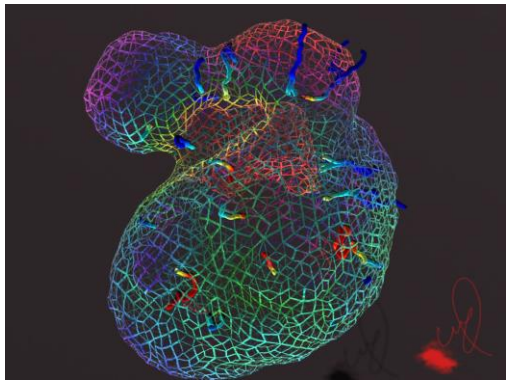
What is Computer Graphics

- 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

Goals of 3D Computer Graphics (cont.)

Goals of 3D Computer Graphics (cont.)

Applications of Computer Graphics

Video Games

Digital Visual Effects (VFX)

Featured Animations

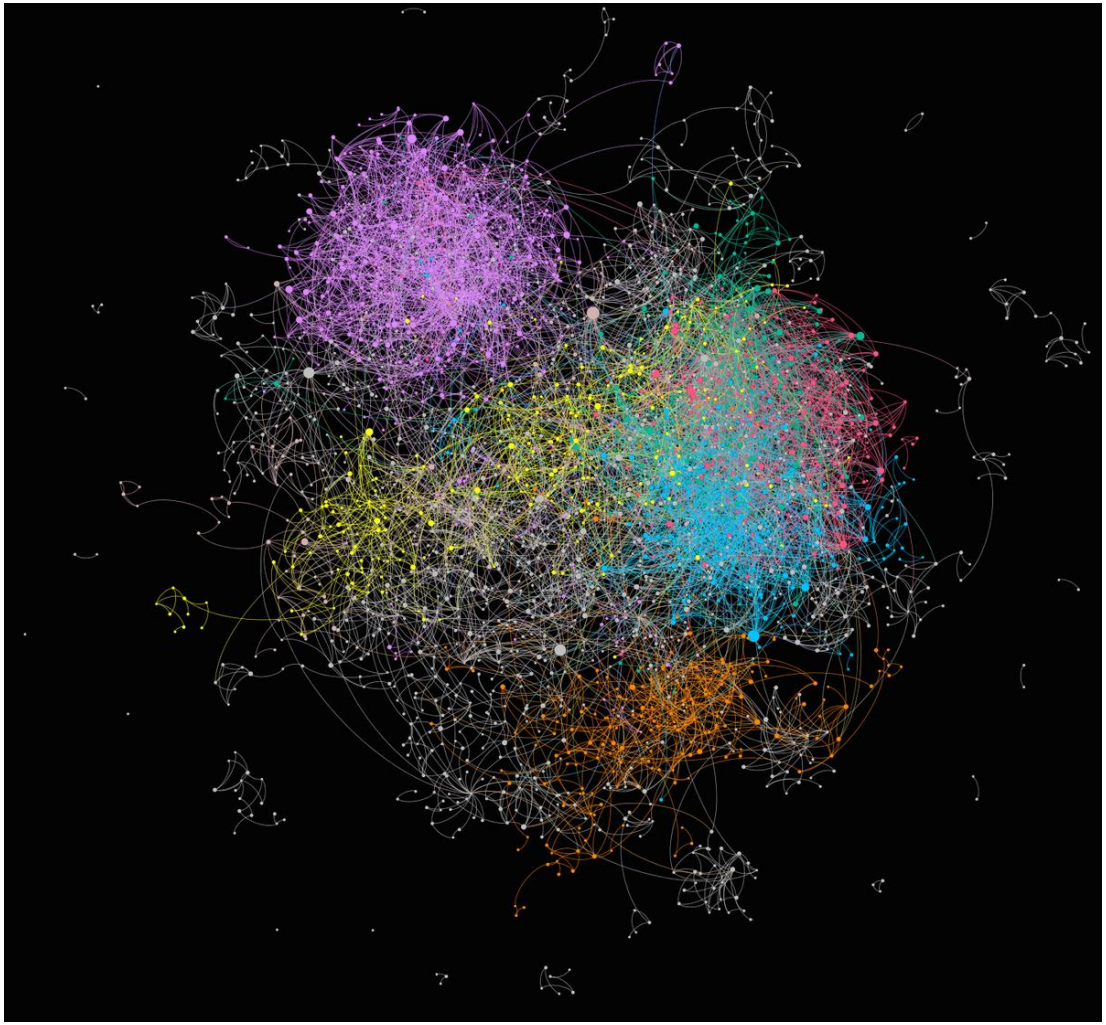
Animes

Virtual Reality (VR)

Augmented and Mixed Reality (AR, MR)

Simulation

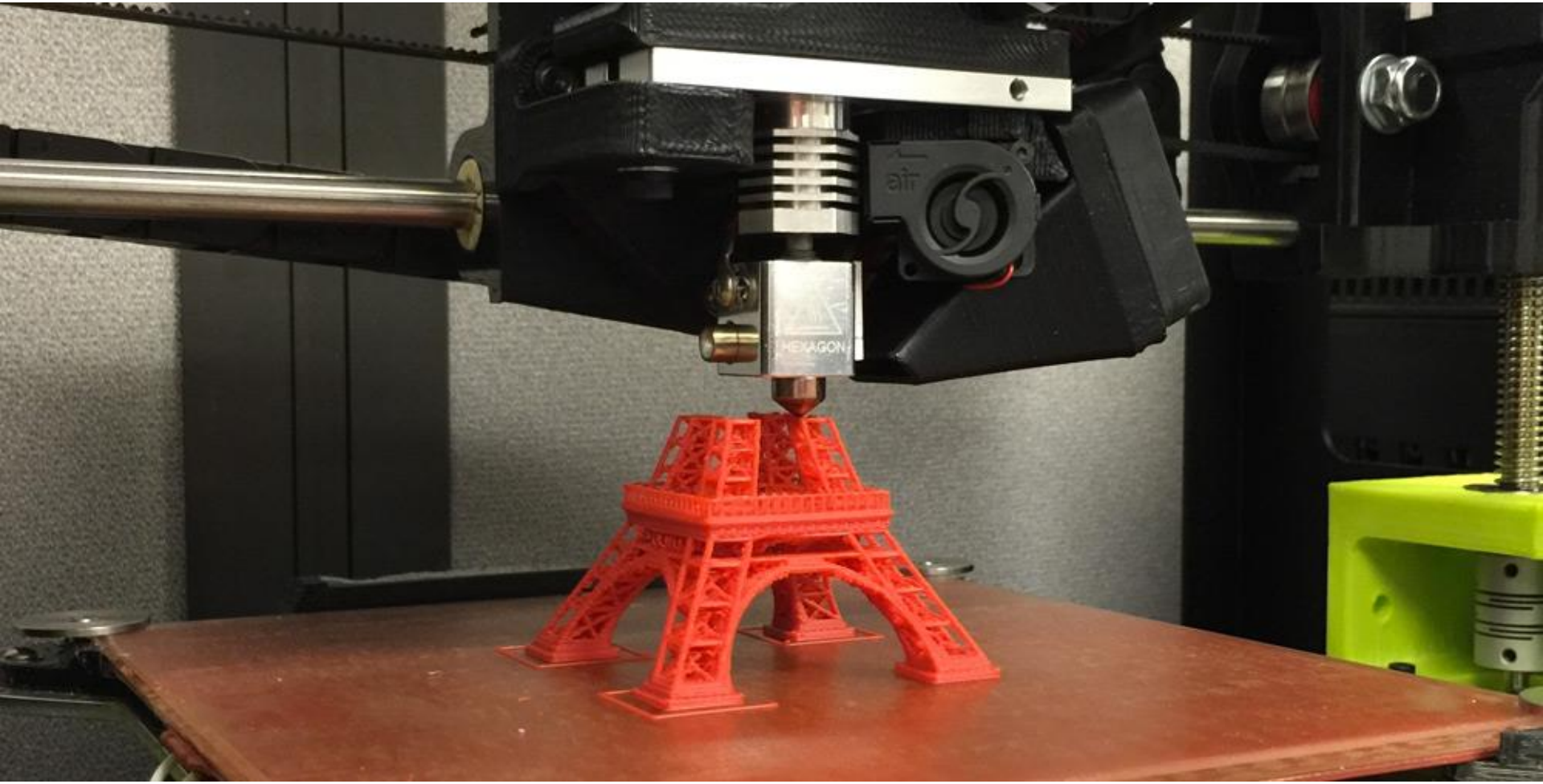
Visualization



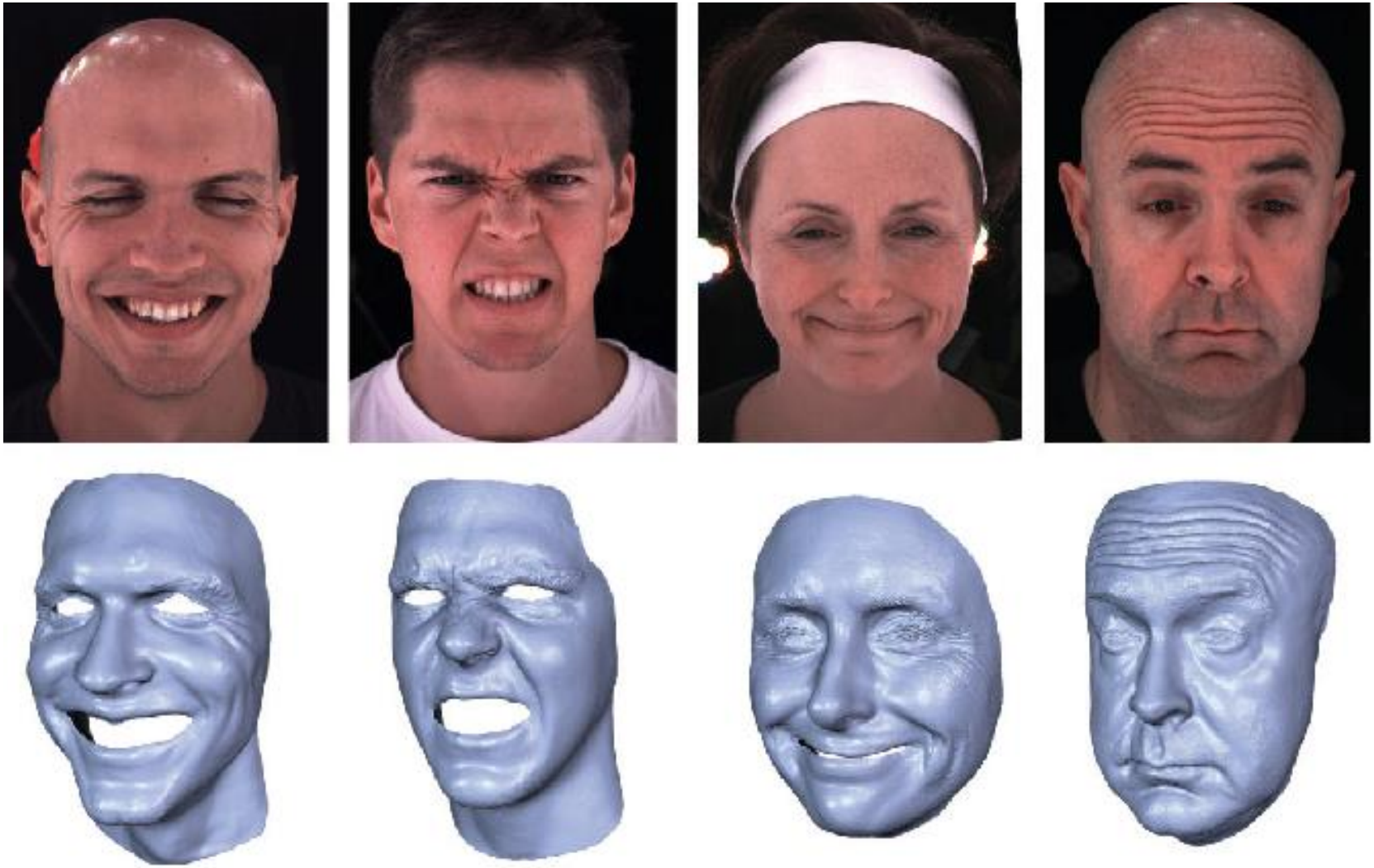
Medical Imaging

Computer-Aided Design

Fabrication

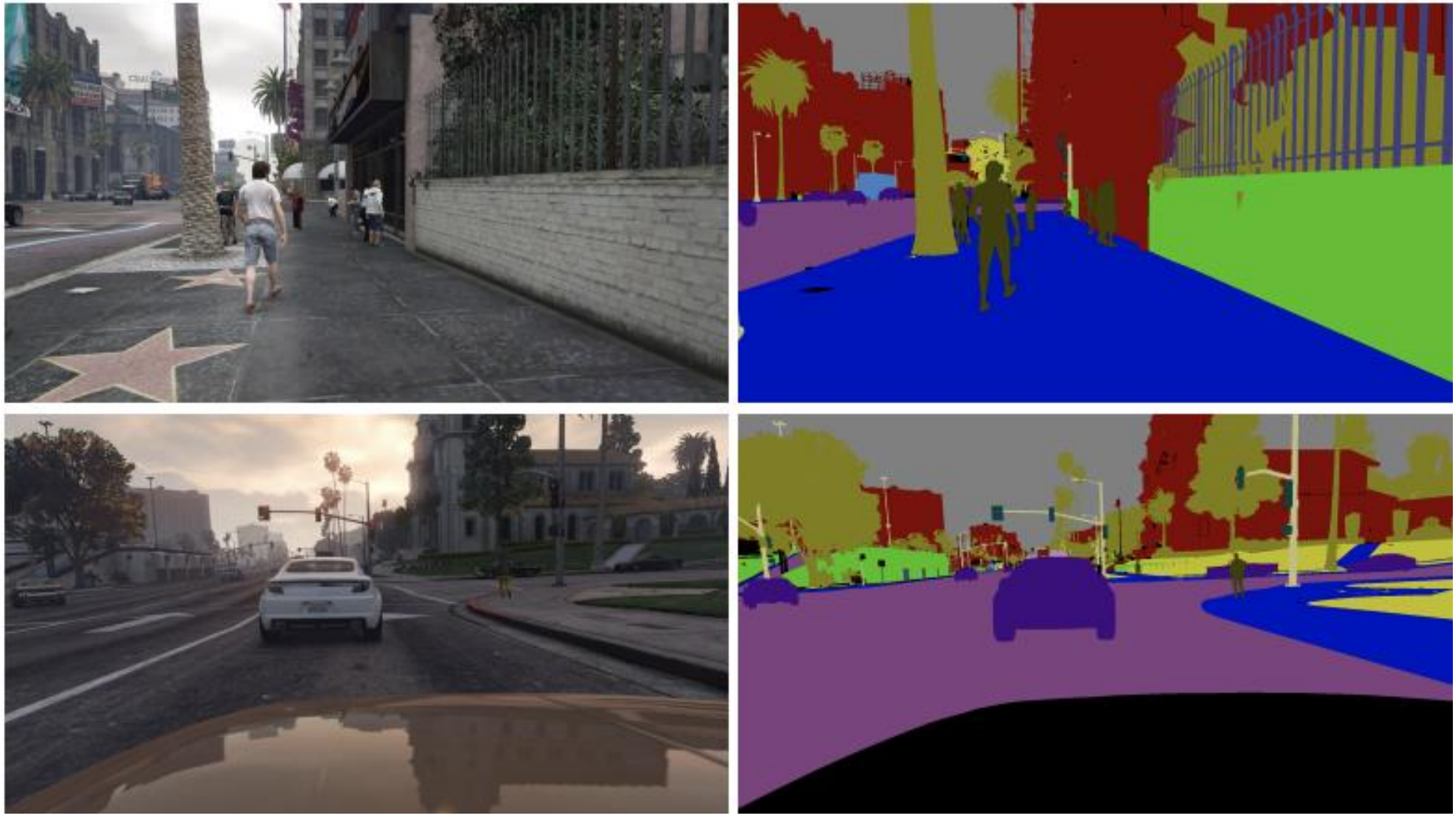


3D Reconstruction



Machine (Deep) Learning

GTA5 Database



A Quick Overview for How to Synthesize an Image

How to Synthesize an Image

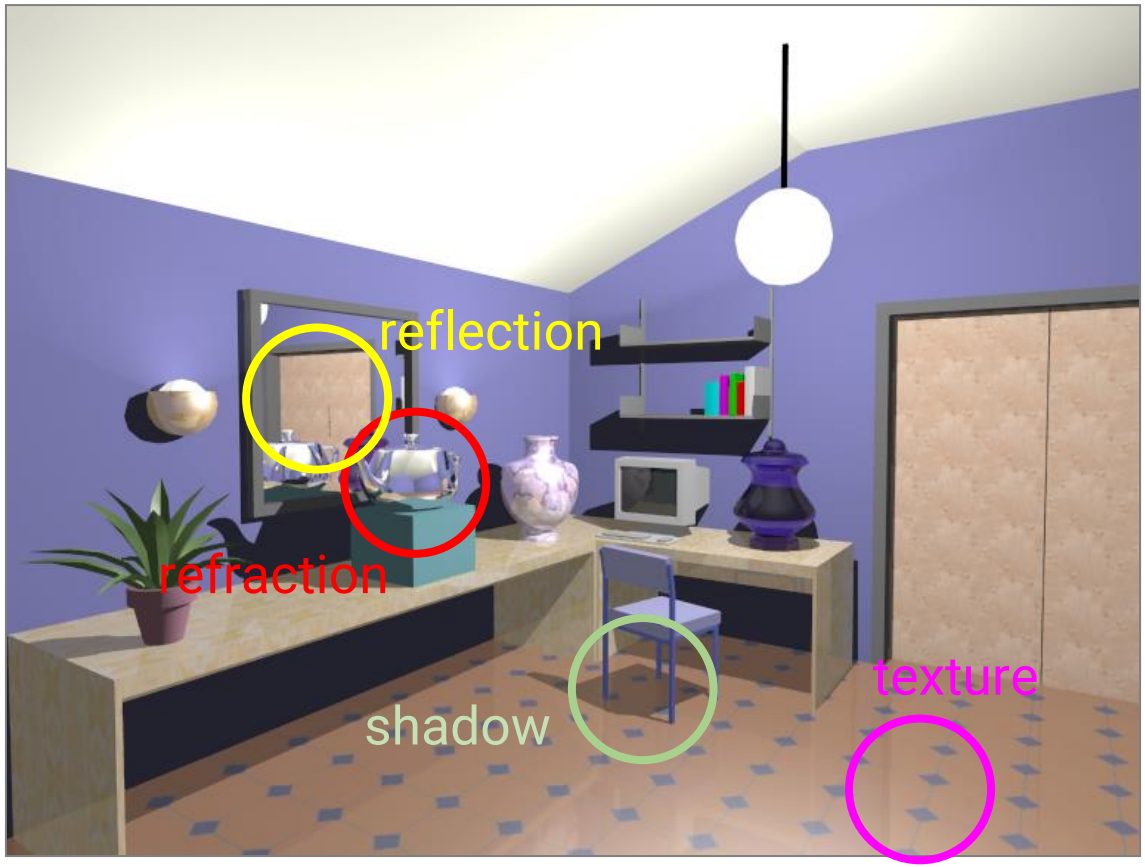
- Model geometry of the 3D objects (scene)

How to Synthesize an Image (cont.)

- Model materials of the 3D objects and simulate lighting

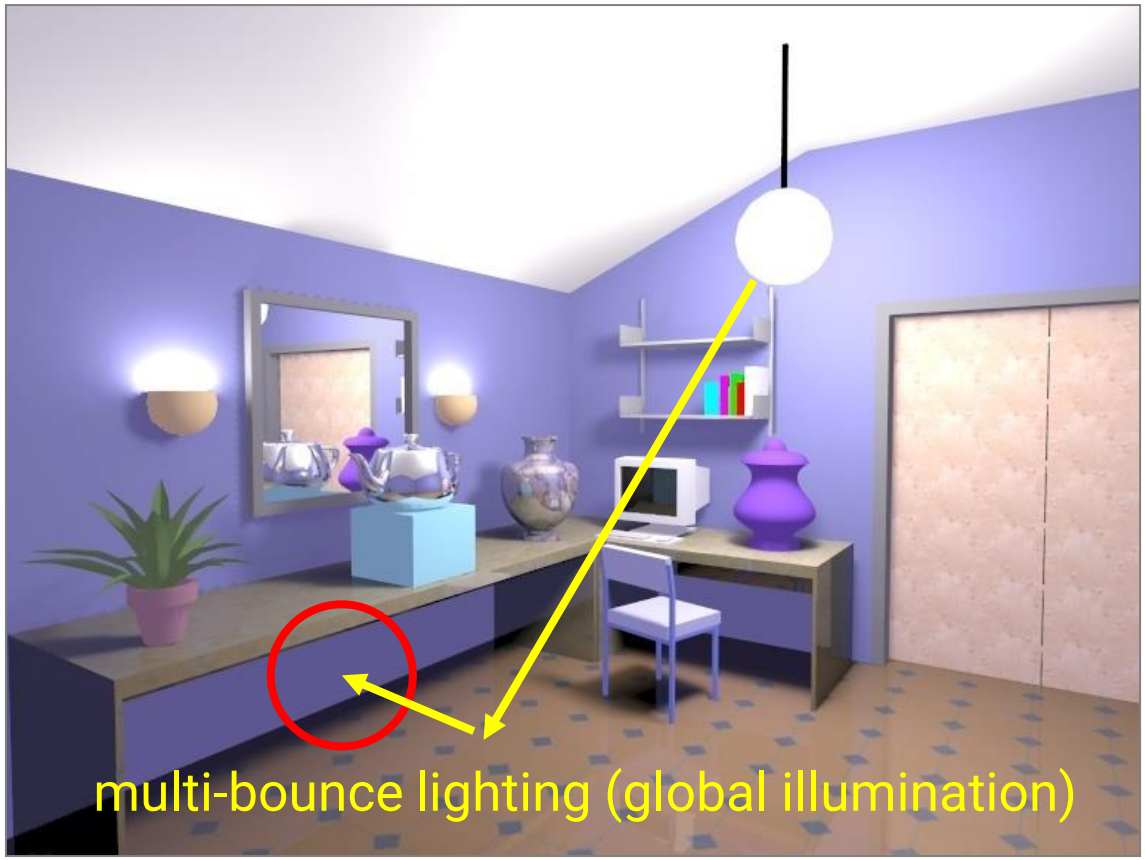
How to Synthesize an Image (cont.)

- Simulate more realistic materials and lighting phenomena



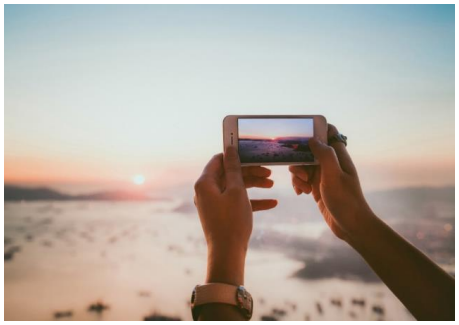
How to Synthesize an Image (cont.)

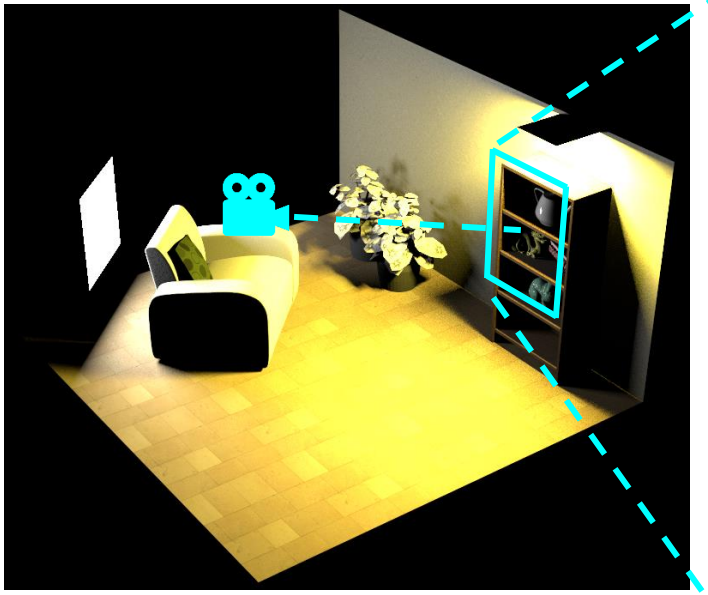
- Simulate more complex light paths



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**

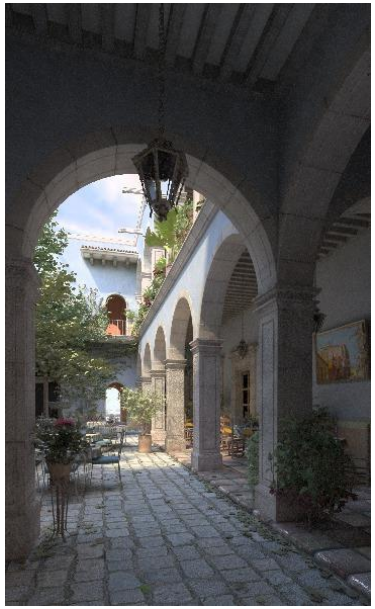




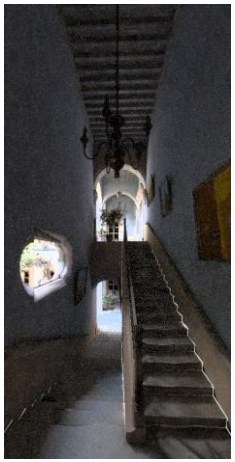
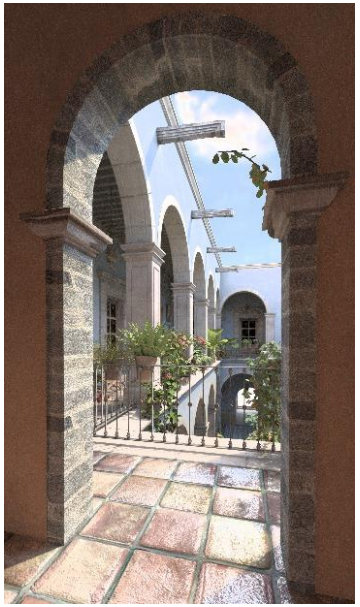
3D virtual world

rendered image

How to Synthesize an Image (cont.)

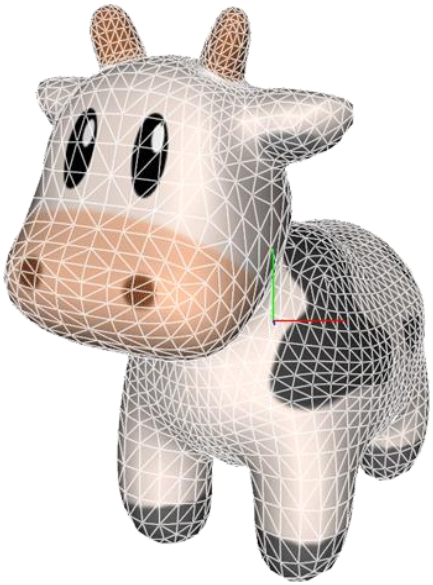




Major Topics of Computer Graphics

Three Pillars of Computer Graphics



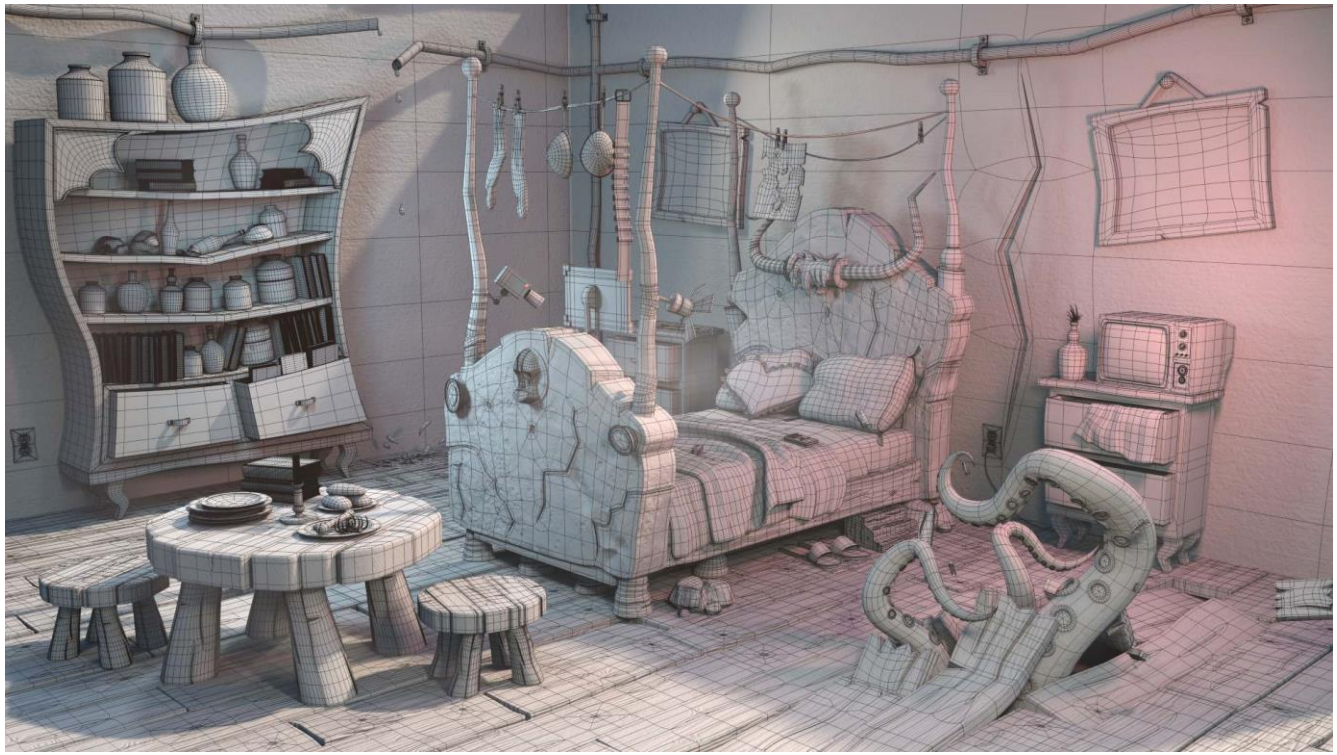
Modeling

Rendering

Animation

Modeling

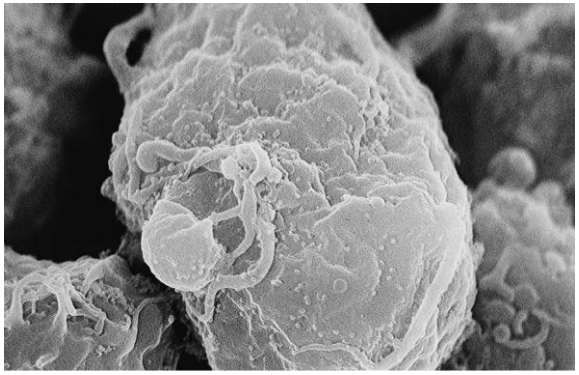
- Build 3D representation of the virtual world
- The process of generating “data” in computer graphics



Modeling (cont.)

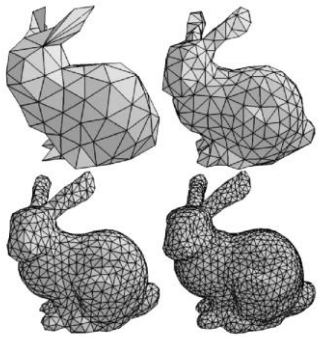
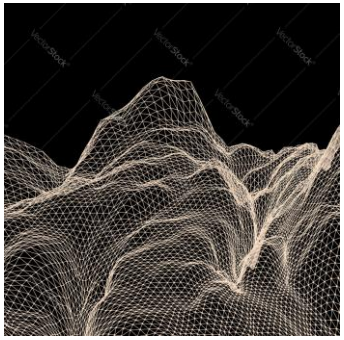
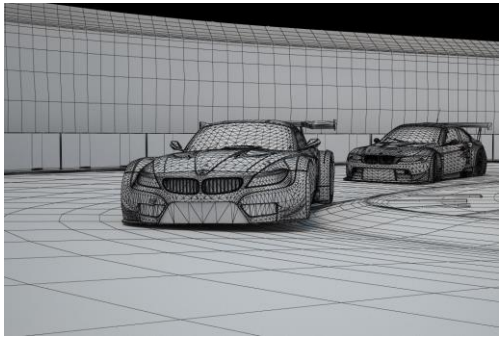
- World geometries are diverse!






Modeling (cont.)

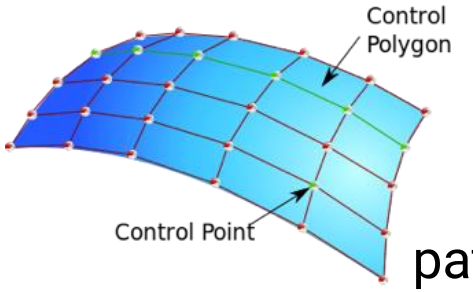
- World geometries are diverse!
- Using different representations including curves, surfaces, volumes

meshes (by polygons, usually triangles)

volume data

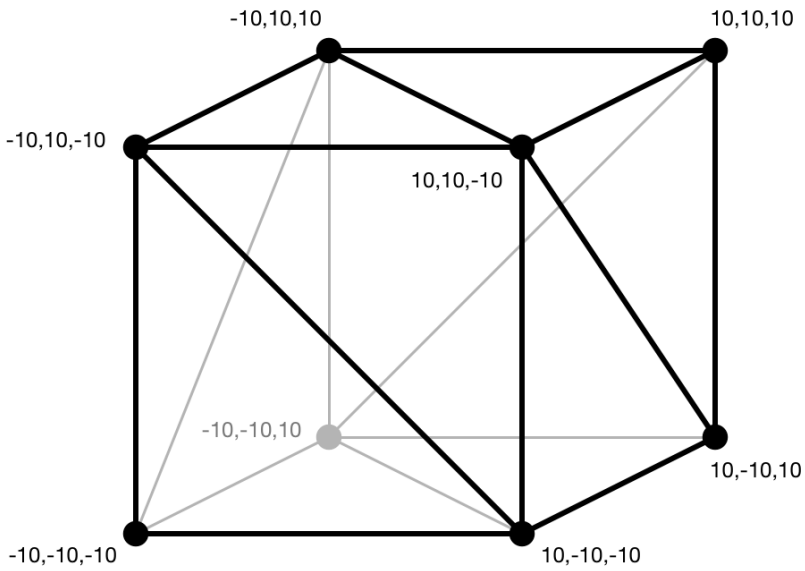
curves



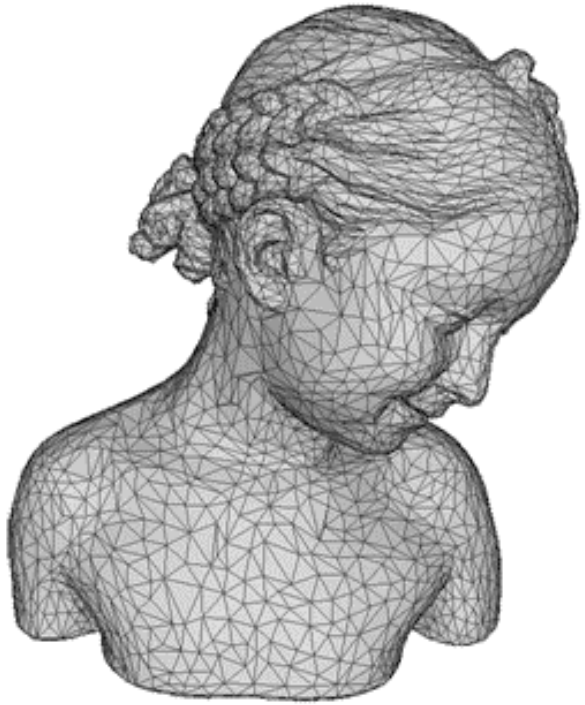
patches

Modeling (cont.)

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



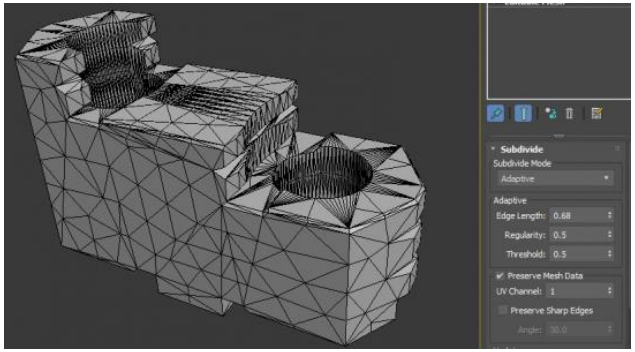
12 triangles



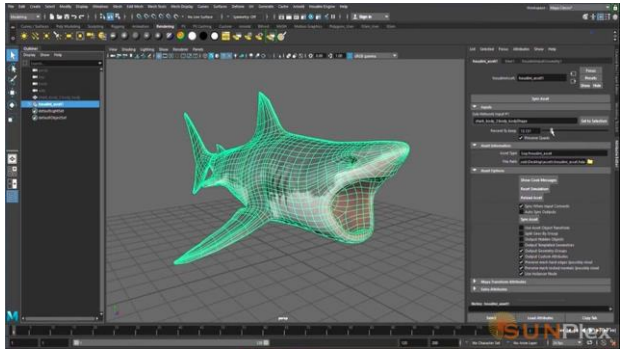
10K triangles

Modeling (cont.)

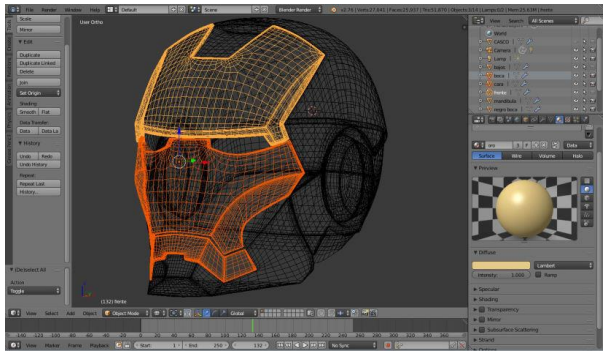
- 3D models are usually obtained by professional manipulations in 3D modeling tools



Blender



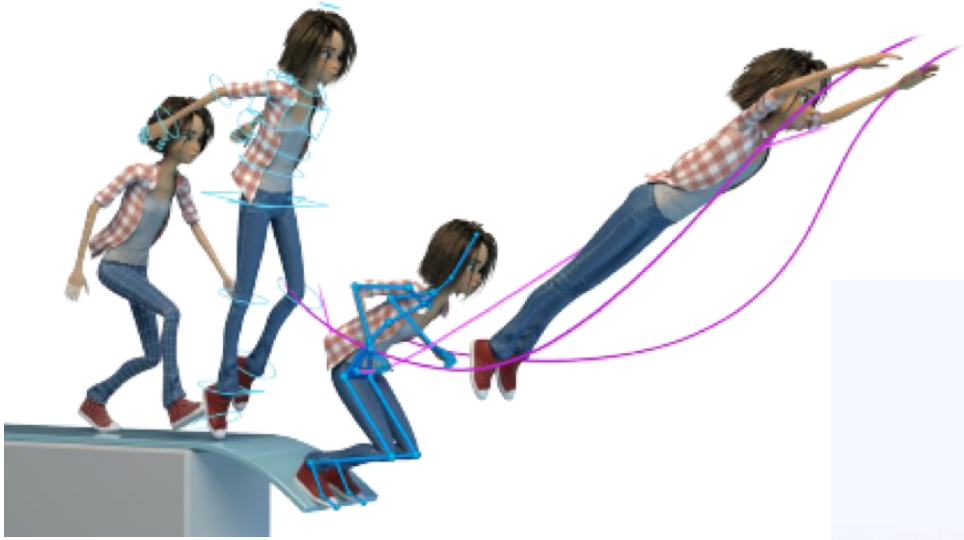
Maya

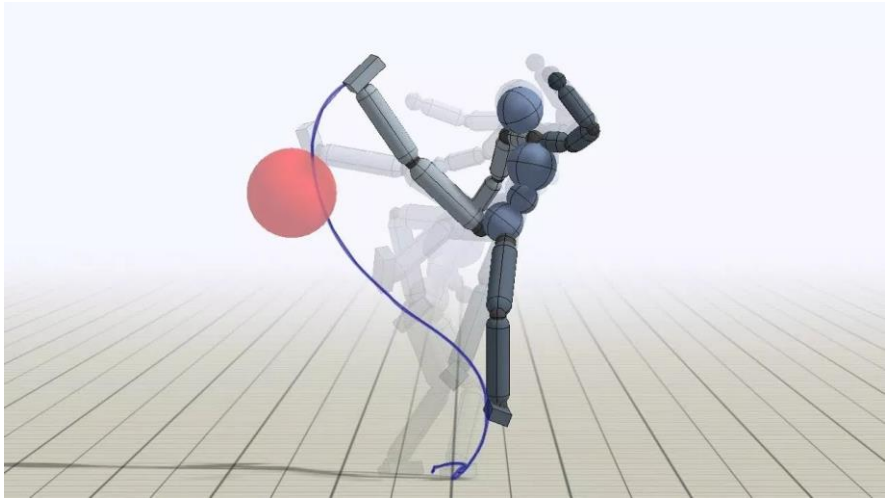


AUTODESK 3DS MAX

Animation

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



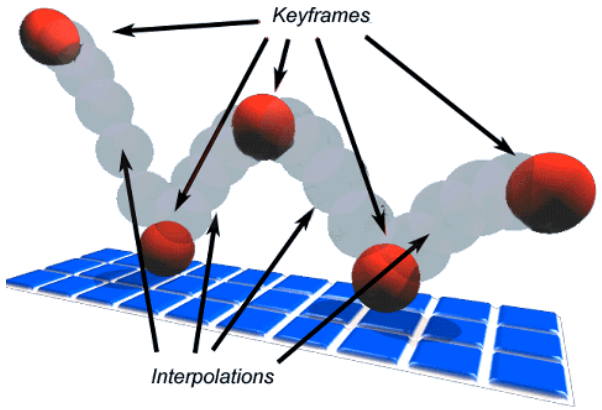


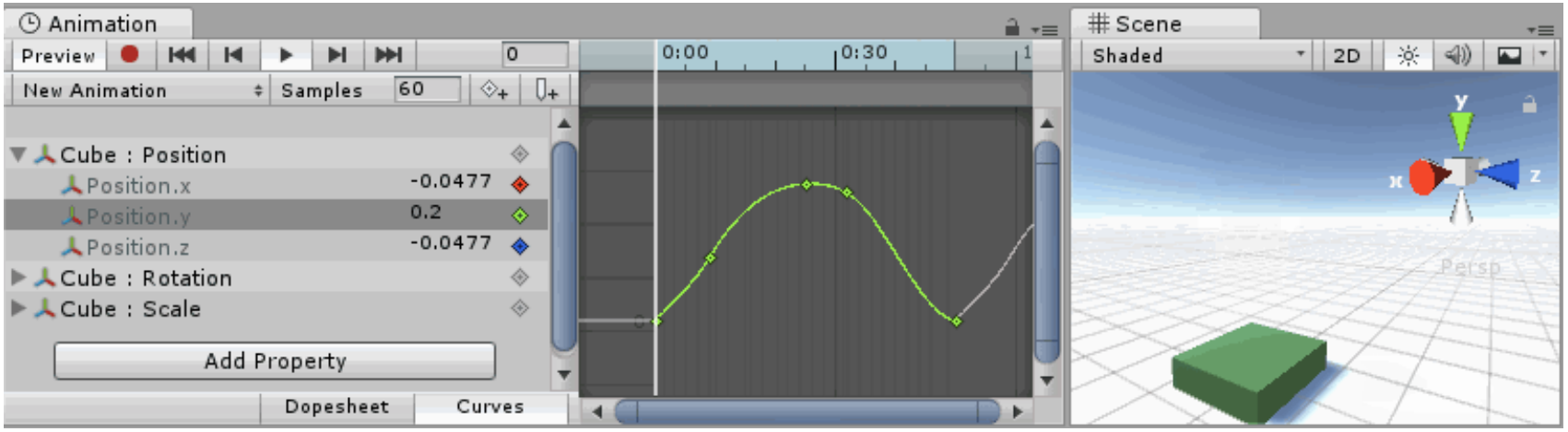
Animation (cont.)

- Animations are usually expected to be physically-based

Animation (cont.)

- Keyframe-based animations



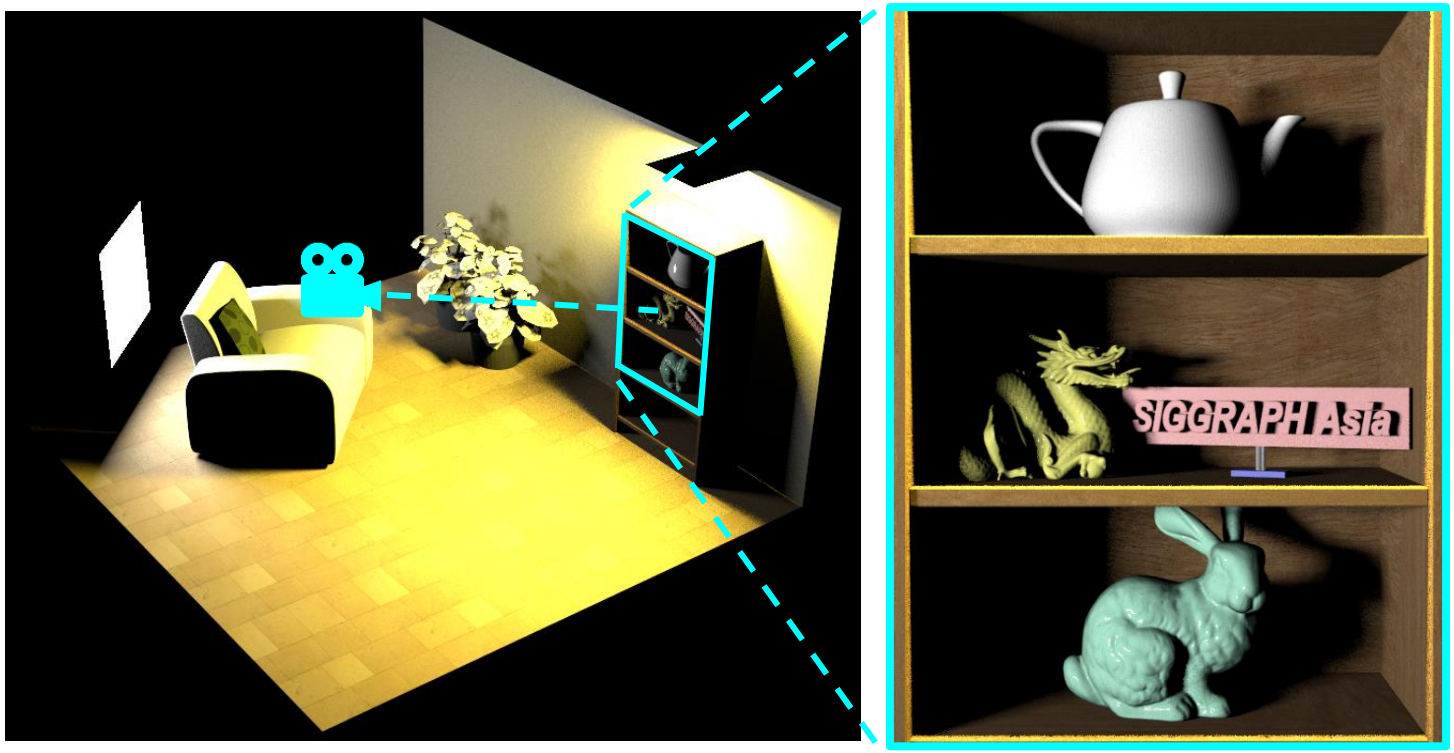


Animation (cont.)

- Motion capture

Rendering

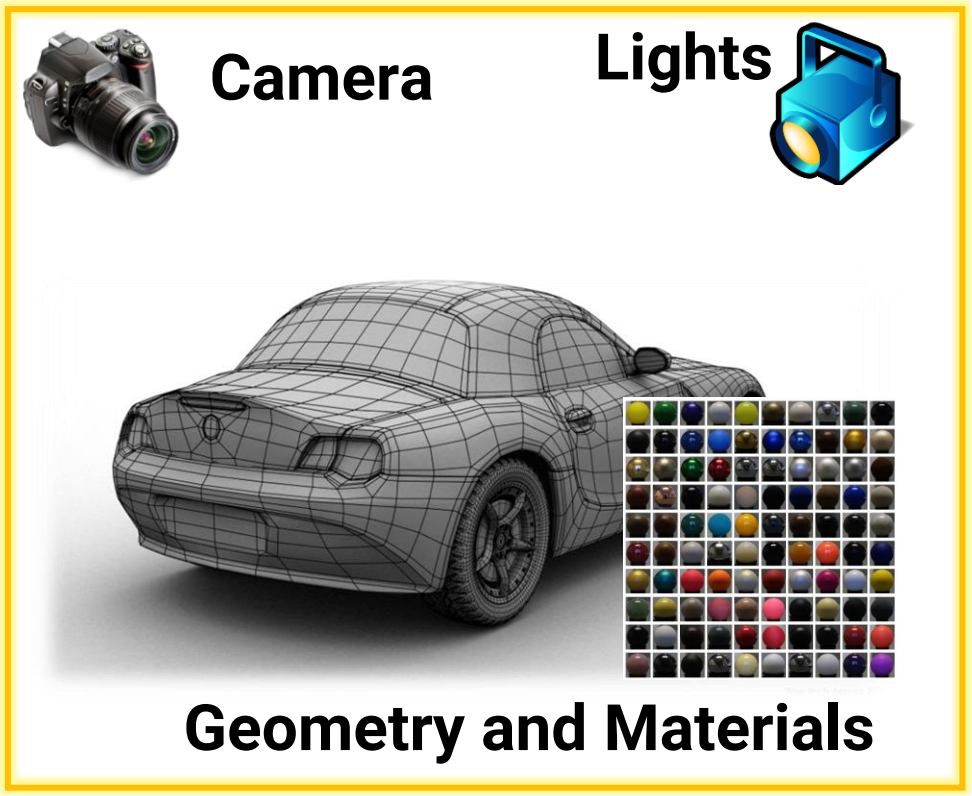
- Simulate the appearance of virtual objects and synthesize the final image



3D virtual world rendered image

Rendering (cont.)

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

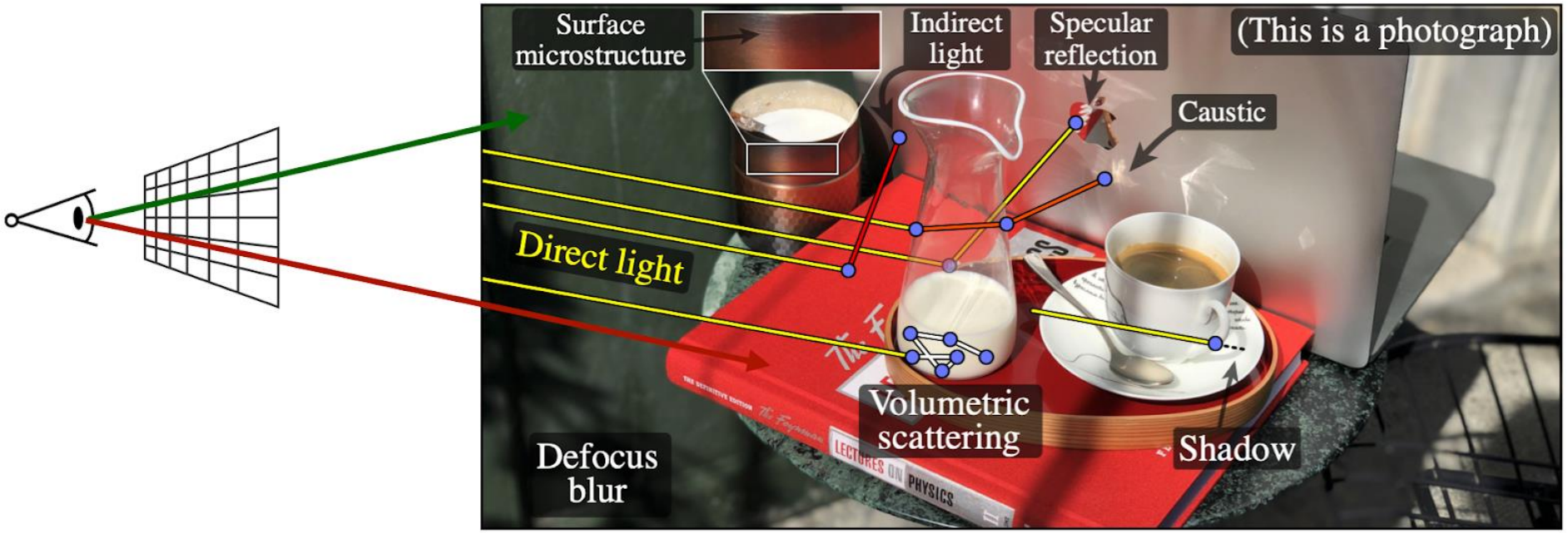


input: 3D description of a scene

output: 2D synthetic image

Rendering (cont.)

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



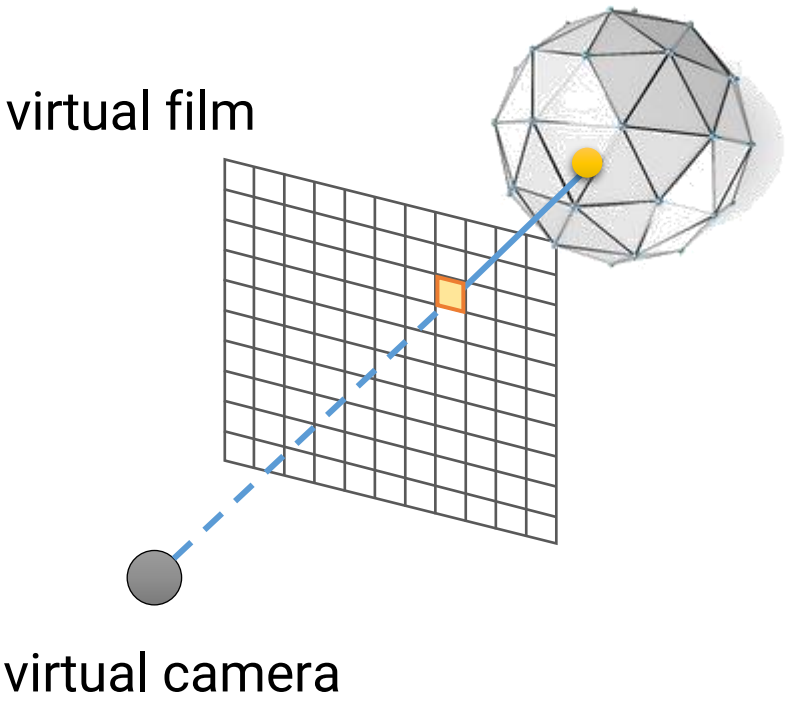
Rendering (cont.)

- Non-photo-realistic rendering

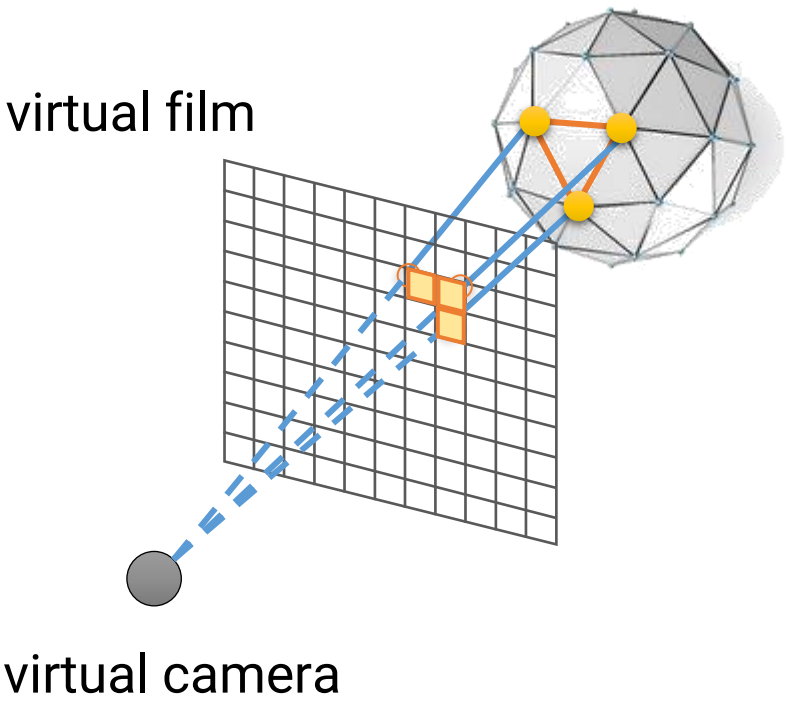
Rendering (cont.)

- 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

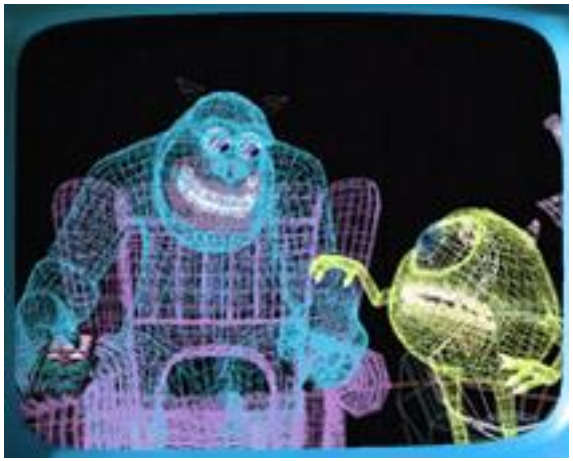
storyboard

voice

storyreel

look and feel

Animation Production Pipeline (cont.)



modeling / articulation

layout

animation

shading / lighting

rendering

final touch

Outline

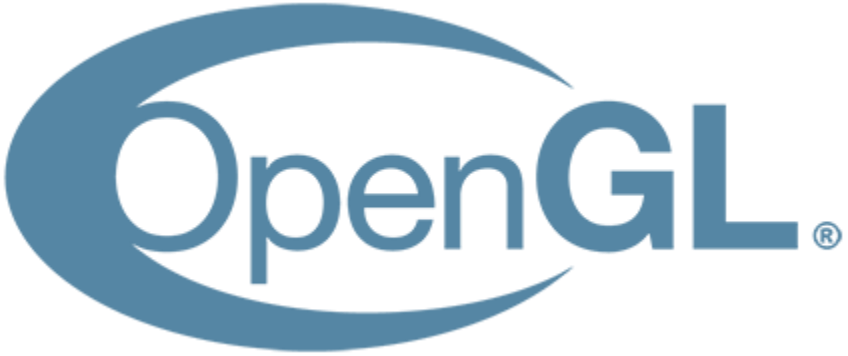
- Course information, policy, and rules
- Introduction to computer graphics
- **Introduction to graphics programming**
- Homework assignments and rendering competition

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

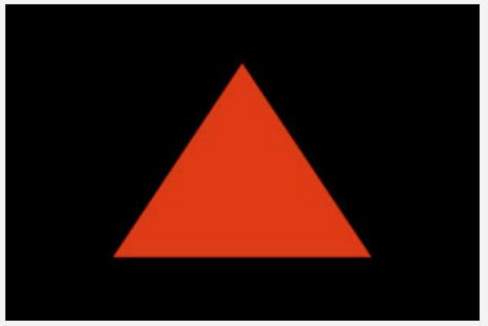
OpenGL

- 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      glClear(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
```

```
#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?

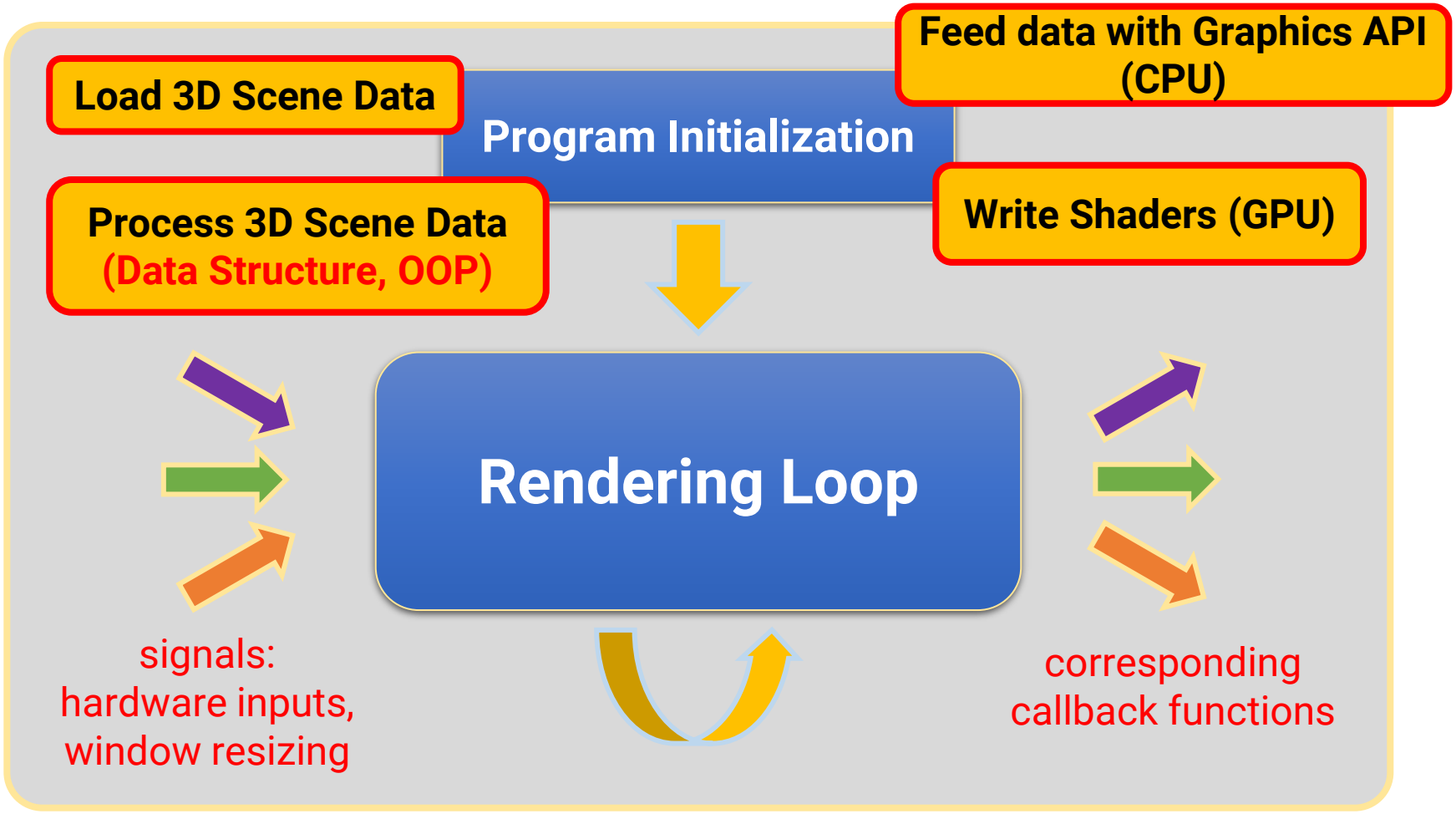
- A simple program to draw a triangle on the screen
  - **457** lines of C++ code

```
    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 pipelineMSCreateInfo = {};
    pipelineMSCreateInfo.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;
```

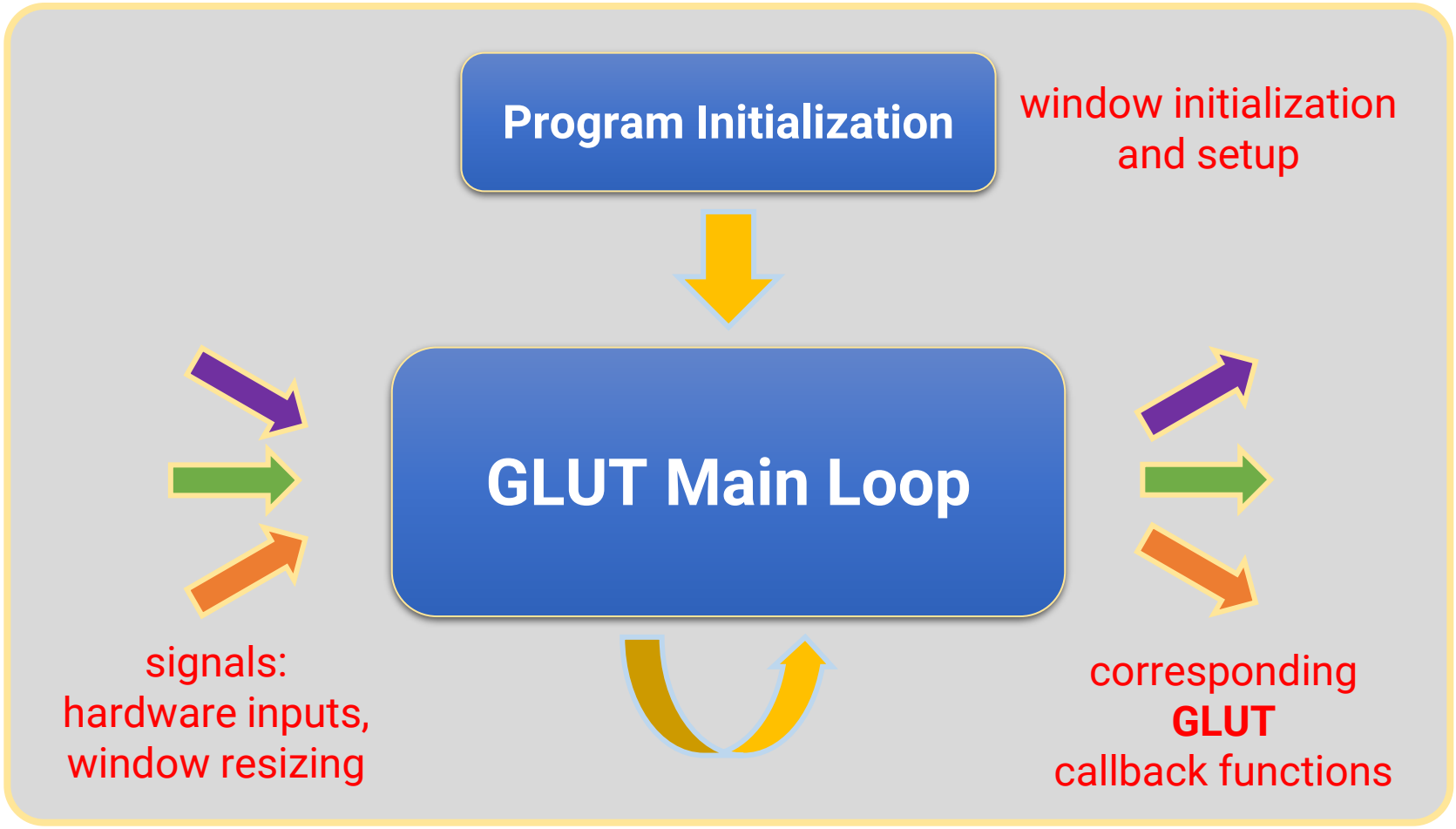
Life Cycle of a Rendering Engine



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 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");

    // Initialization.
    SetupRenderState();

    // Register callback functions.
    glutDisplayFunc(RenderSceneCB);
    glutIdleFunc(RenderSceneCB);
    glutReshapeFunc(ReshapeCB);
    glutSpecialFunc(ProcessSpecialKeysCB);
    glutKeyboardFunc(ProcessKeysCB);

    // Start rendering loop.
    glutMainLoop();

    return 0;
}
```

create the window and set window properties

do initialization jobs

register callback functions

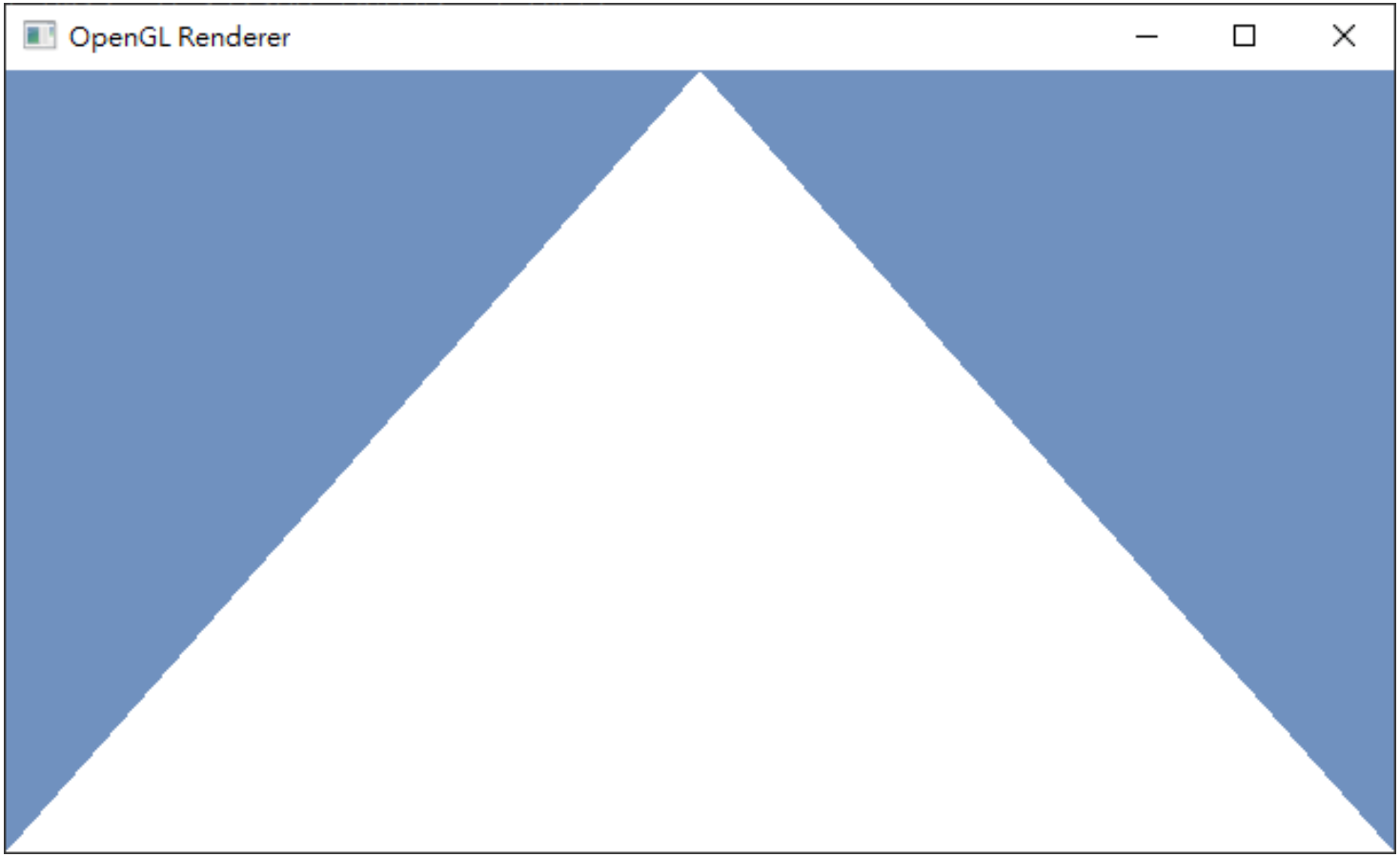
start the main loop

FreeGLUT Window

- FreeGLUT will create and maintain a window on screen

Next Two Weeks

- We will learn how to render a single triangle



Outline

- Course information, policy, and rules
- Introduction to computer graphics
- Introduction to graphics programming
- **Homework assignments and rendering competition**

Topics We Plan to Cover

Basic

HW1
- Geometry Representation
- Transformations
- Camera

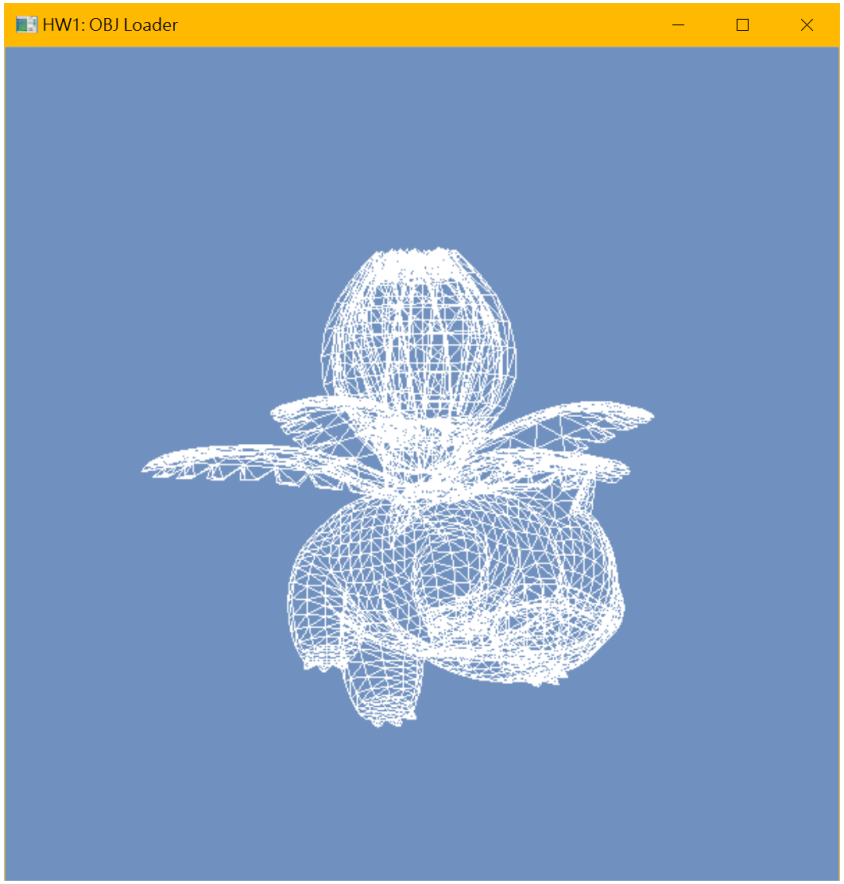
HW2
- GPU Graphics Pipeline
- Shading

HW3
- Textures
- Skybox

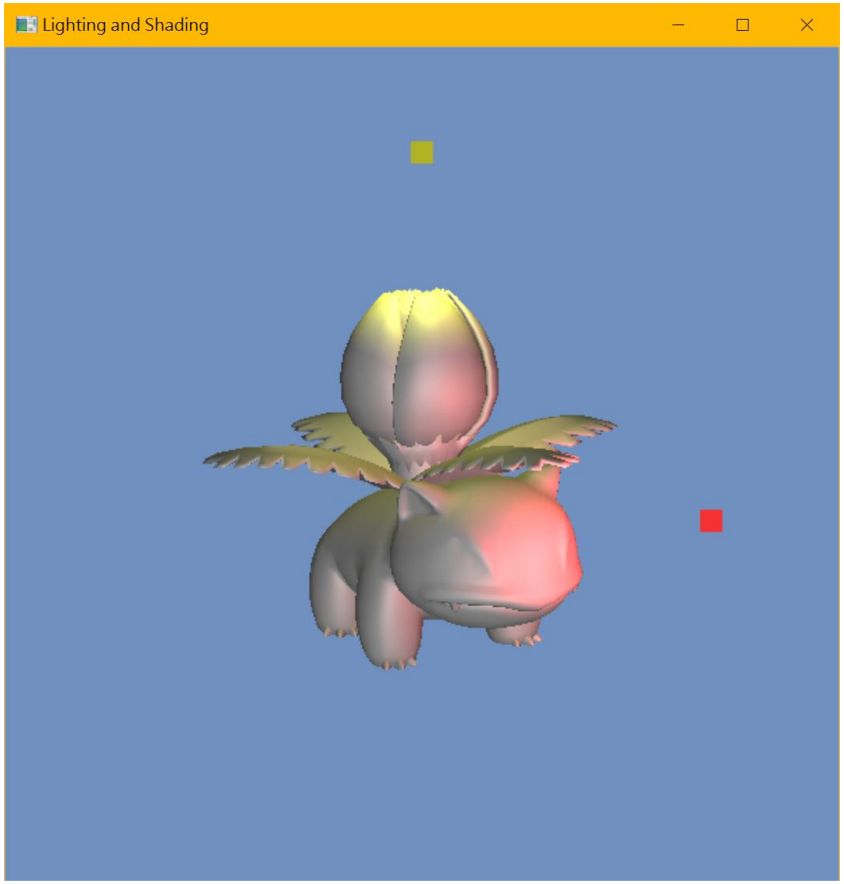
Advanced

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

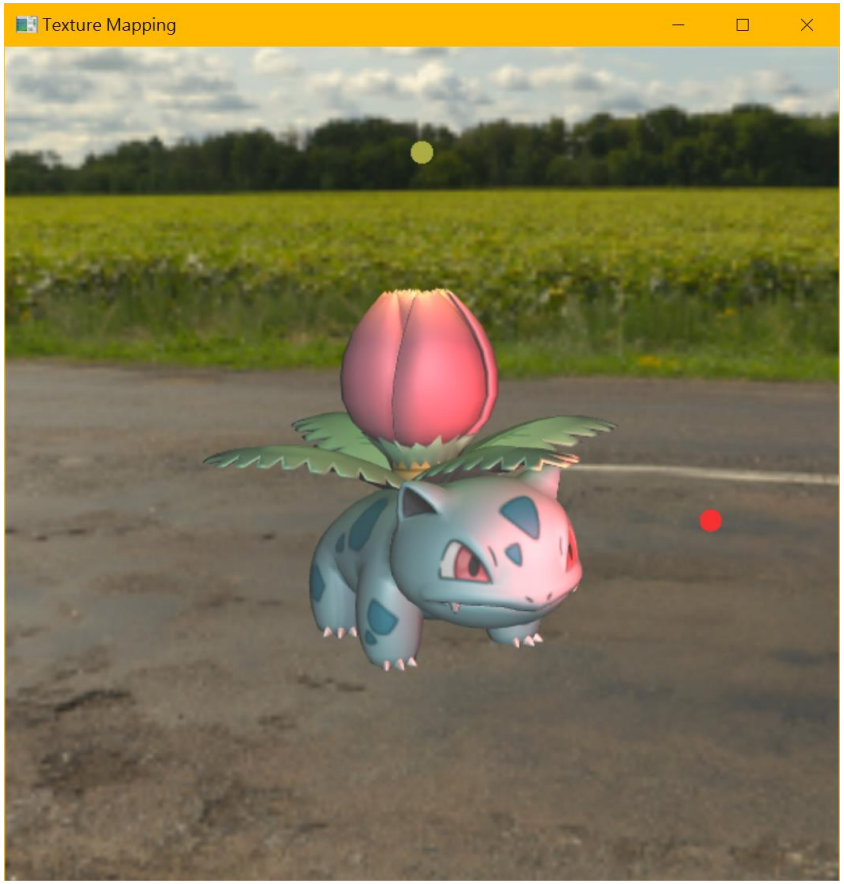
HW1: Geometry Representation (18%)

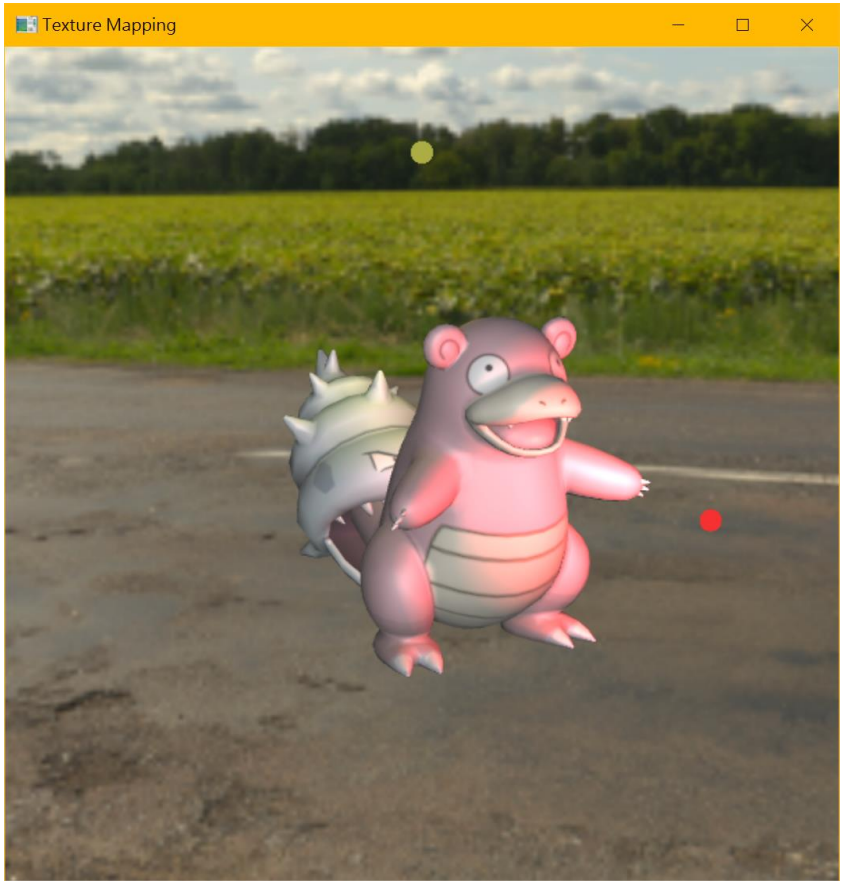


HW2: Lighting and Shading (18%)



HW3: Texturing and Skybox (9%)





Rendering Competition (5%)

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

Rendering Competition (5%)


END OF PRESENTATION
ANY QUESTIONS?