University of Colorado Boulder

Introduction to Virtual Reality

VR Design Patterns

Professor **Dan Szafir**

Computer Science & ATLAS Institute
University of Colorado Boulder

How do we develop VR applications?

VR Development Platforms

- Code
  - OpenGL
  - DirectX
- Editor
- Code + Editor
  - Unity
  - Unreal Engine
  - CryEngine
  - Amazon Lumberyard
- Web
  - WebGL
  - Three.js
  - WebVR

Native Development

Greatest flexibility and control

Long development times

Used for

- Extremely high performance applications
- Building your own framework

Game Engines

Software development framework that makes it easier to develop games

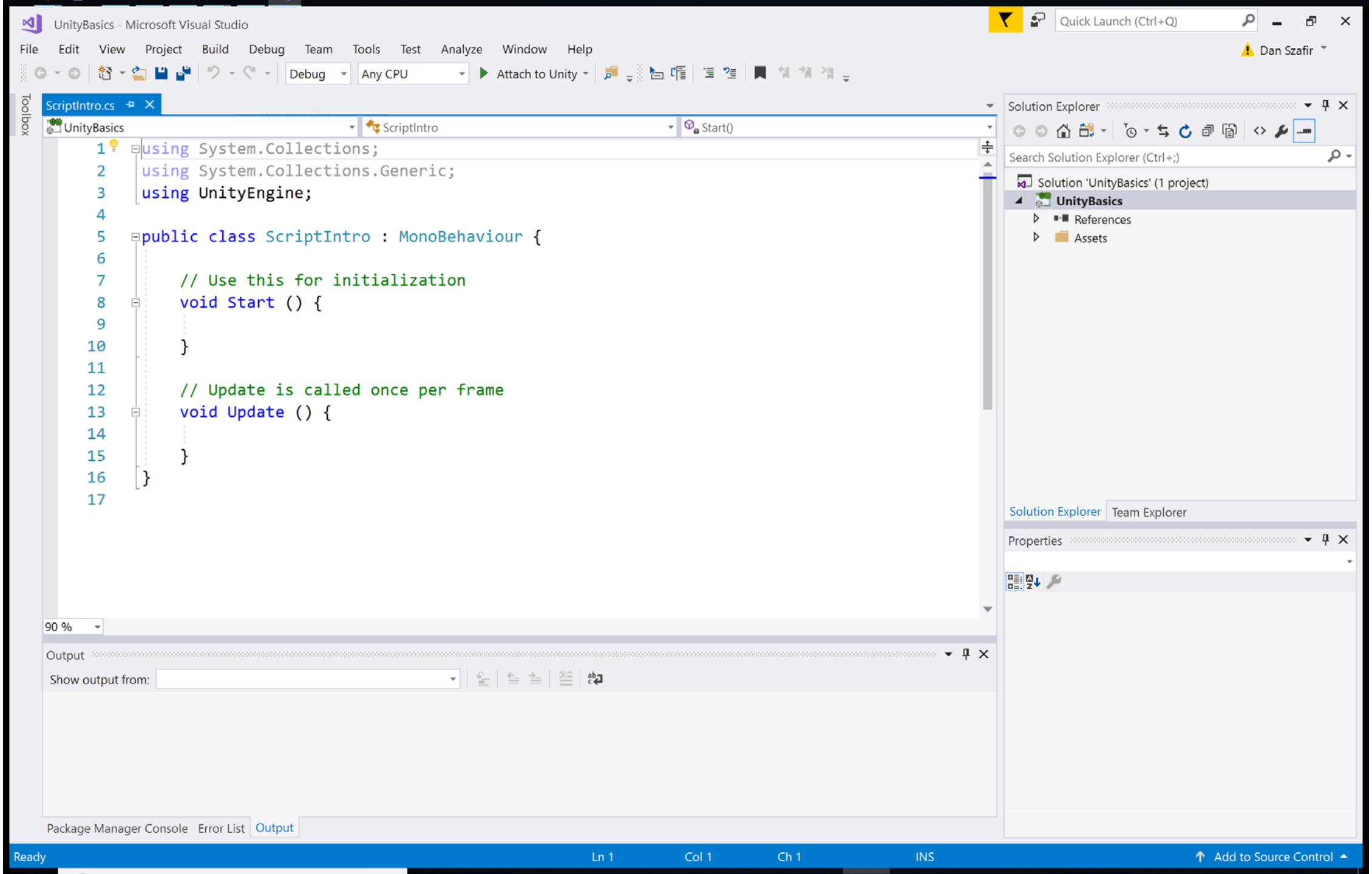
- Games = interactive experiences and/or simulations

Goal: reduce iteration and development time

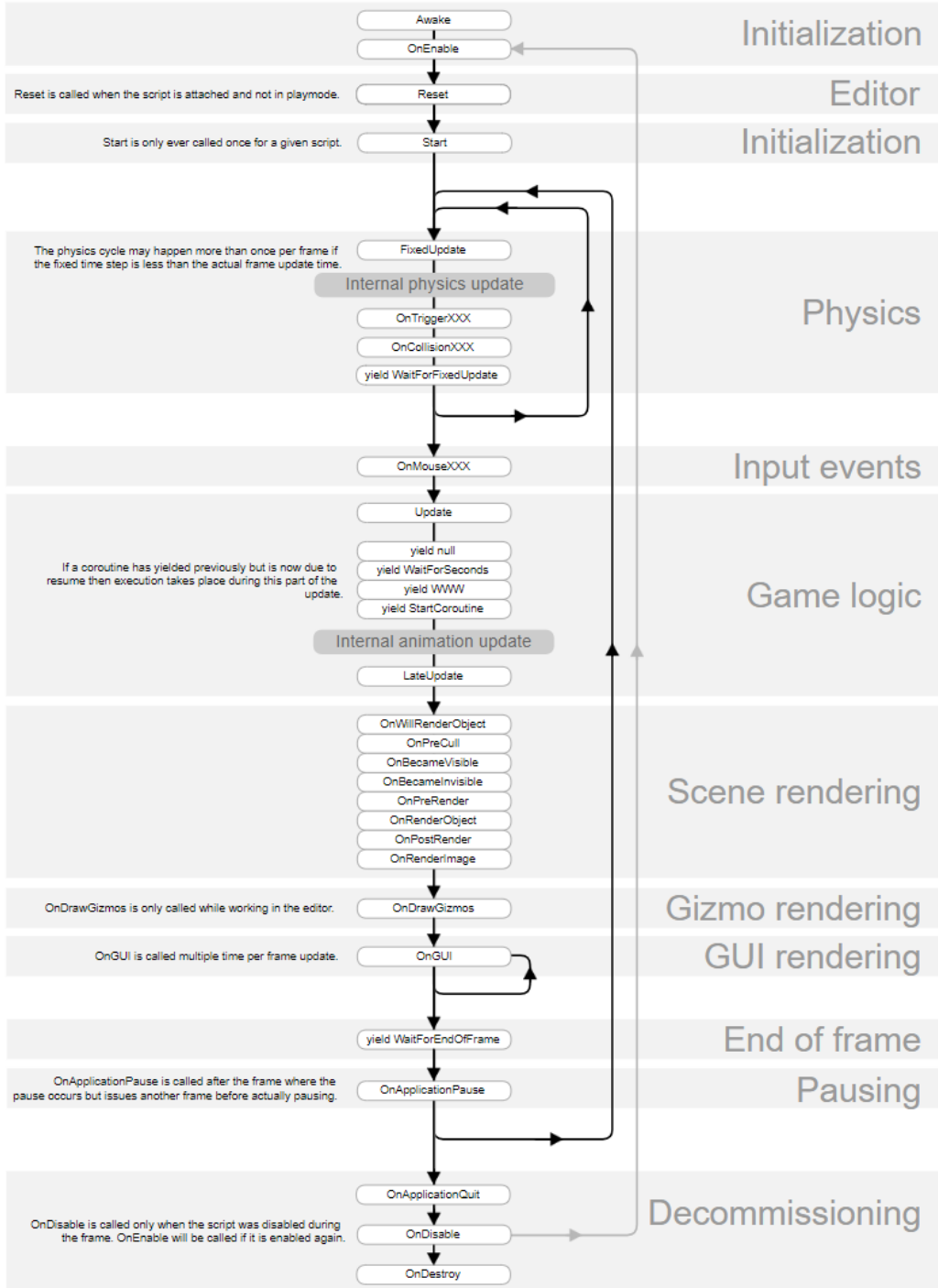
Provides

- 3D rendering
- Physics
- Sound
- Scripting
- Animation
- Asset management
- …

Unity Scripting Basics


UnityBasics - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Dan Szafir
Debug
Any CPU
Attach to Unity
Toolbox
ScriptIntro.cs
UnityBasics
ScriptIntro
Start()
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class ScriptIntro : MonoBehaviour {
6
7 // Use this for initialization
8 void Start () {
9
10 }
11
12 // Update is called once per frame
13 void Update () {
14
15 }
16 }
17
90 %
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'UnityBasics' (1 project)
UnityBasics
References
Assets
Solution Explorer
Team Explorer
Properties
Output
Show output from:
Package Manager Console
Error List
Output
Ready
Ln 1
Col 1
Ch 1
INS
Add to Source Control

Where do these functions come from?


Initialization
Awake
OnEnable
Editor
Reset is called when the script is attached and not in playmode.
Reset
Initialization
Start is only ever called once for a given script.
Start
Physics
The physics cycle may happen more than once per frame if the fixed time step is less than the actual frame update time.
FixedUpdate
Internal physics update
OnTriggerXXX
OnCollisionXXX
yield WaitForFixedUpdate
Input events
OnMouseXXX
Game logic
Update
If a coroutine has yielded previously but is now due to resume then execution takes place during this part of the update.
yield null
yield WaitForSeconds
yield WWW
yield StartCoroutine
Internal animation update
LateUpdate
Scene rendering
OnWillRenderObject
OnPreCull
OnBecameVisible
OnBecameInvisible
OnPreRender
OnRenderObject
OnPostRender
OnRenderImage
Gizmo rendering
OnDrawGizmos is only called while working in the editor.
OnDrawGizmos
GUI rendering
OnGUI is called multiple time per frame update.
OnGUI
End of frame
yield WaitForEndOfFrame
Pausing
OnApplicationPause is called after the frame where the pause occurs but issues another frame before actually pausing.
OnApplicationPause
Decommissioning
OnApplicationQuit
OnDisable is called only when the script was disabled during the frame. OnEnable will be called if it is enabled again.
OnDisable
OnDestroy

Special Unity Functions Invoked by the Unity Engine

Awake()
- Executed once as the scene loads when the component awakens (e.g., just after a prefab is instantiated)

Start()
- Executed once just before the first frame update

Update()
- Invoked once per frame during play mode

FixedUpdate()
- Called on a reliable timer; may be called multiple times per frame (if low frame rate) or not at all between frames (if frame rate is high)

Many others
- See https://docs.unity3d.com/Manual/ExecutionOrder.html

Miscellaneous Utility

Debugging:

- Debug.Log(“text”)

Timing:

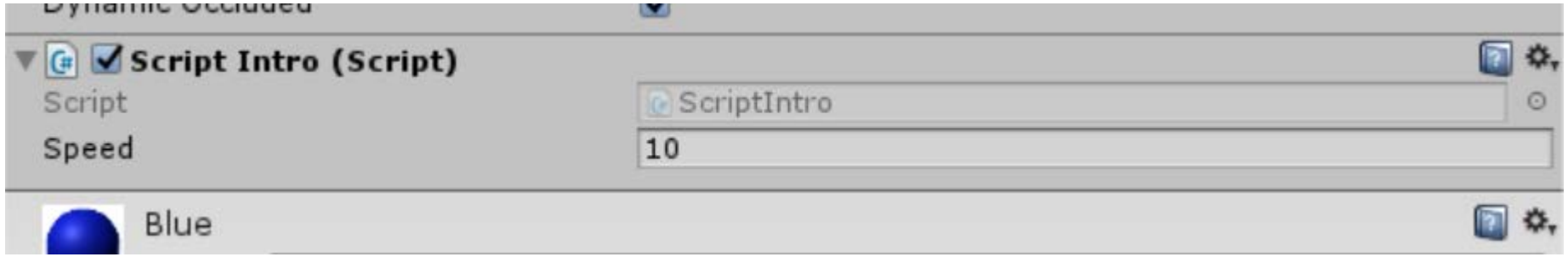
- Time.[x]
  - e.g., Time.time, Time.deltaTime, Time.fixedDeltaTime, Time.timeSinceLevelLoad
- DateTime object
  - DateTime time = DateTime.Now;
  - Time.Hour, time.Minute, time.Second
- TimeSpan object
  - TimeSpan time = DateTime.Now.TimeOfDay;
  - Time.TotalHours, time.TotalMinutes, time.TotalSeconds

Variable Visibility

Declare variables *private* for use just in this script

Declare variables *public* to be visible in the Unity editor (and to other scripts)

e.g., public int speed = 10;



Transform

Position, rotation, and scale of an object

Why is this called “transform”?

e.g., transform.Rotate(Vector3.up, speed * Time.deltaTime);

Can use to instantiate prefabs

```
public prefab somePrefab;
...
Instantiate(somePrefab);
```

Getting References to Other Objects

Main camera: Camera.main

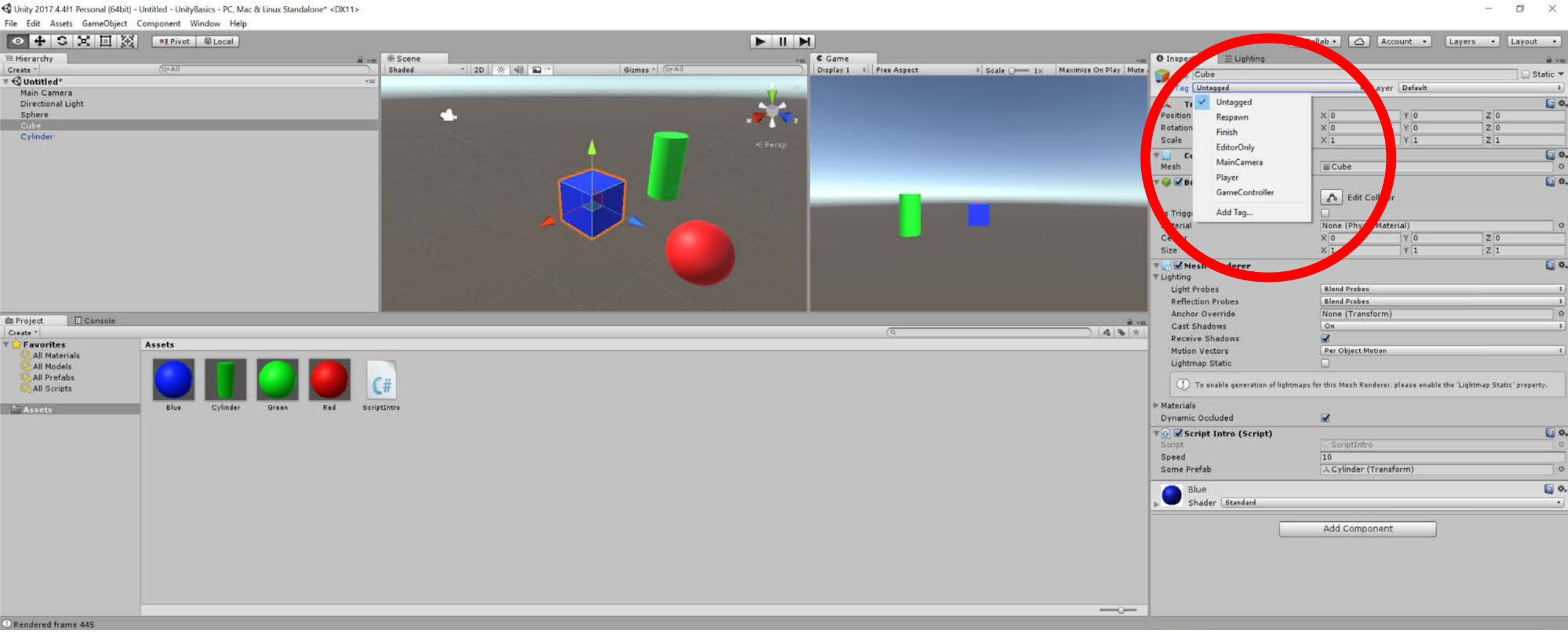
Another GameObject in the scene:

- GameObject.Find(“name”)
- GameObject.FindObjectOfType(Type t)
- GameObject. FindGameObjectWithTag(“tag”)
- …etc. (can also find multiple objects)

Example

- GameObject.Find("Sphere").GetComponent<Renderer>().material .SetColor("_Color", Color.gray);

Tags


Unity 2017.4.4f1 Personal (64bit) - Untitled - UnityBasics - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
Untitled*
Main Camera
Directional Light
Sphere
Cube
Cylinder
Scene
Game
Project
Console
Favorites
All Materials
All Models
All Prefabs
All Scripts
Assets
Blue
Cylinder
Green
Red
ScriptIntro
Inspector
Lighting
Tag
Untagged
Respawn
Finish
EditorOnly
MainCamera
Player
GameController
Add Tag...
Position
Rotation
Scale
Mesh
Light Probes
Reflection Probes
Anchor Override
Cast Shadows
Receive Shadows
Motion Vectors
Lightmap Static
Materials
Dynamic Occluded
Script Intro (Script)
Speed
Some Prefab
Shader
Standard
Add Component
Rendered frame 445

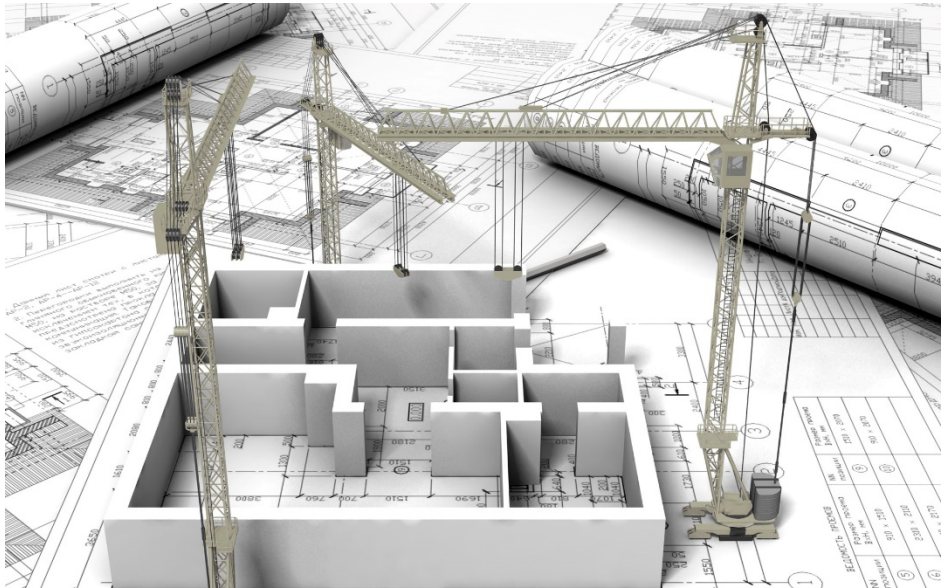
Ok, but how does all this really work?

System Architecture

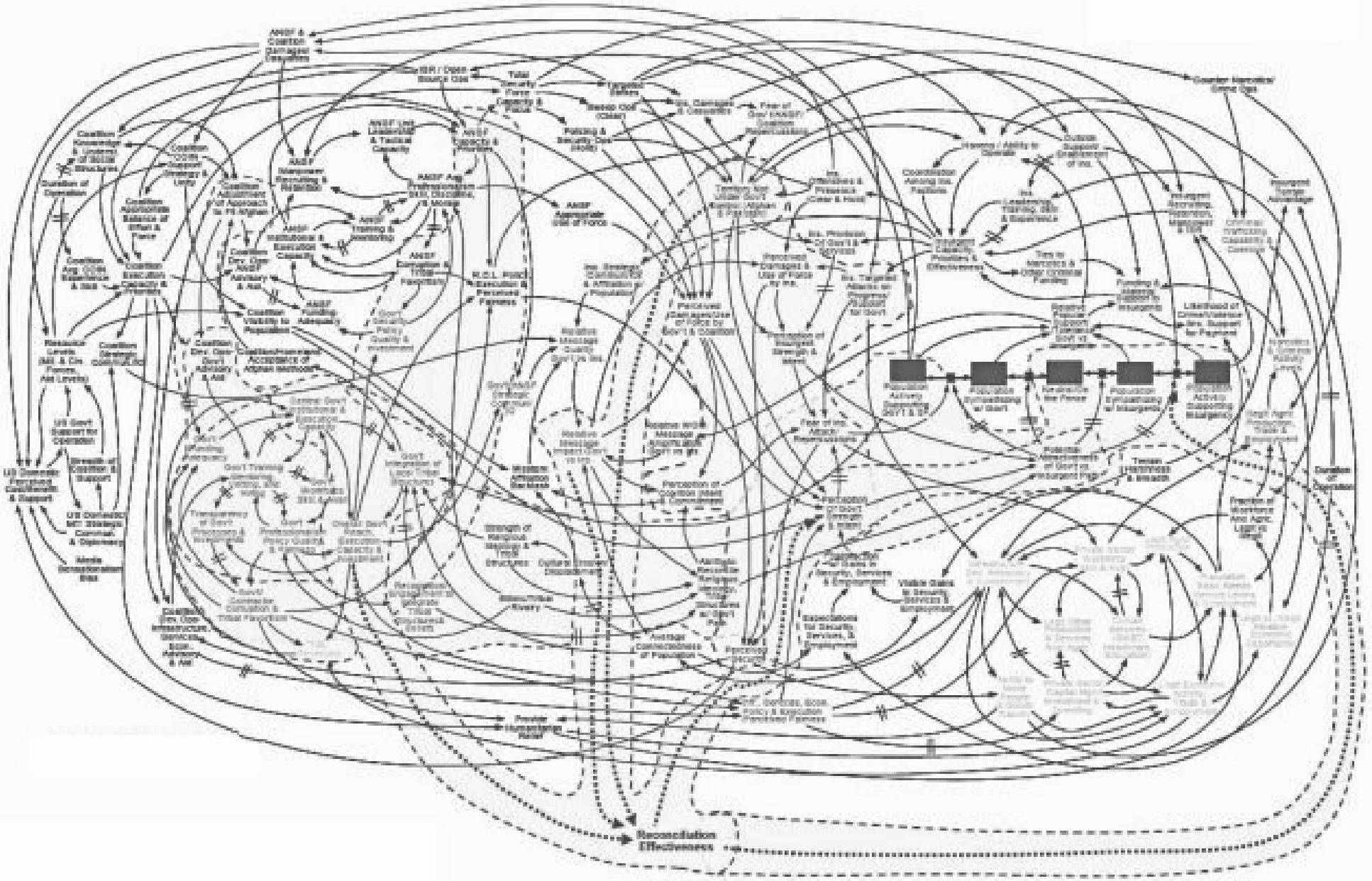
What might be the components of a VR application?

Why might thinking about application architecture be important?

How might we organize system components in a meaningful way?



GOTO E SPAGHETTI CODE



Design Patterns

A “general, repeatable solution to a commonly occurring problem”

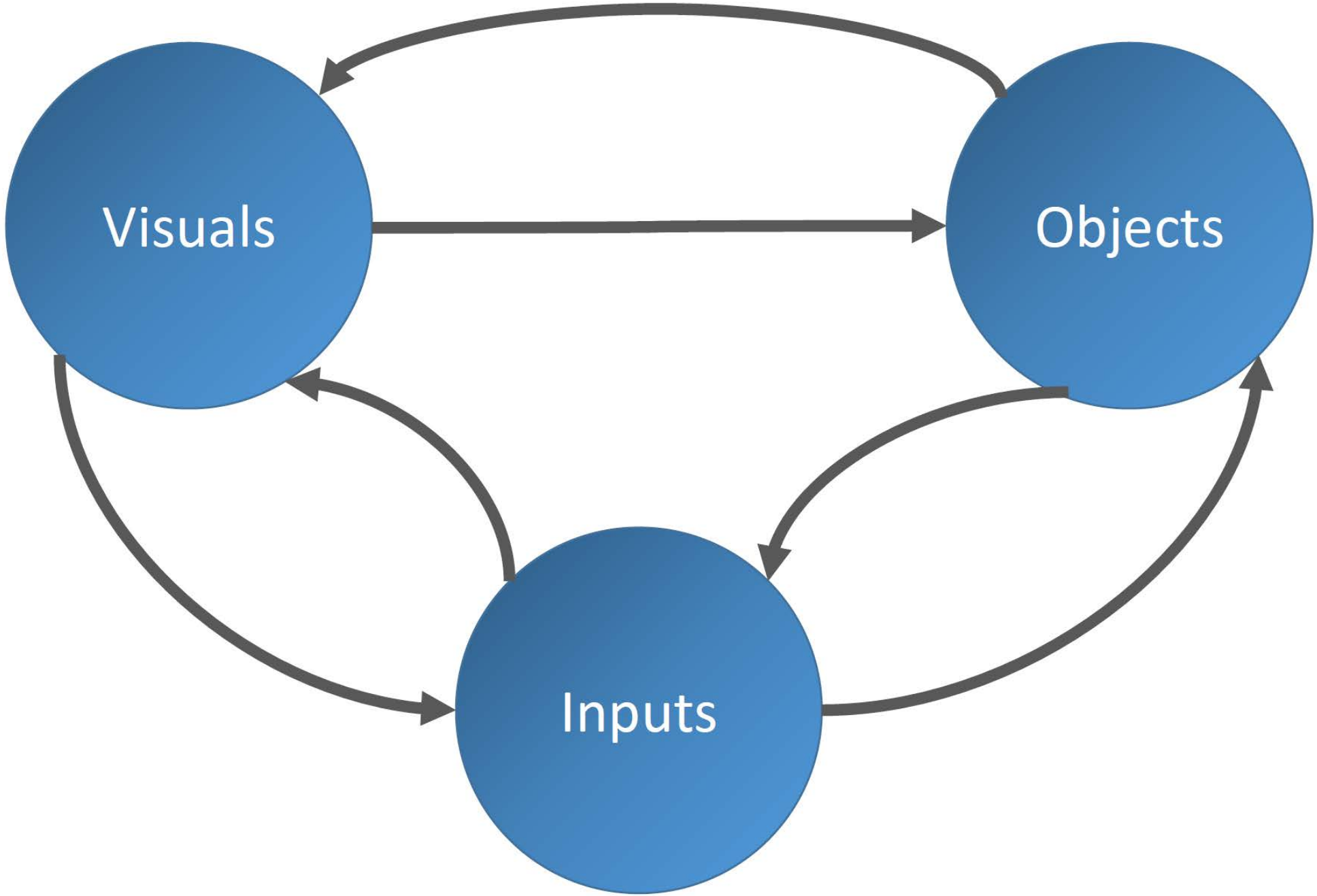
A template, not a finished low-level program

Allows common structure for code, provides high-level logical flow, enables code re-use

In VR, we are concerned with

- Enabling interaction
- Efficiency and speed

Model-View-Controller

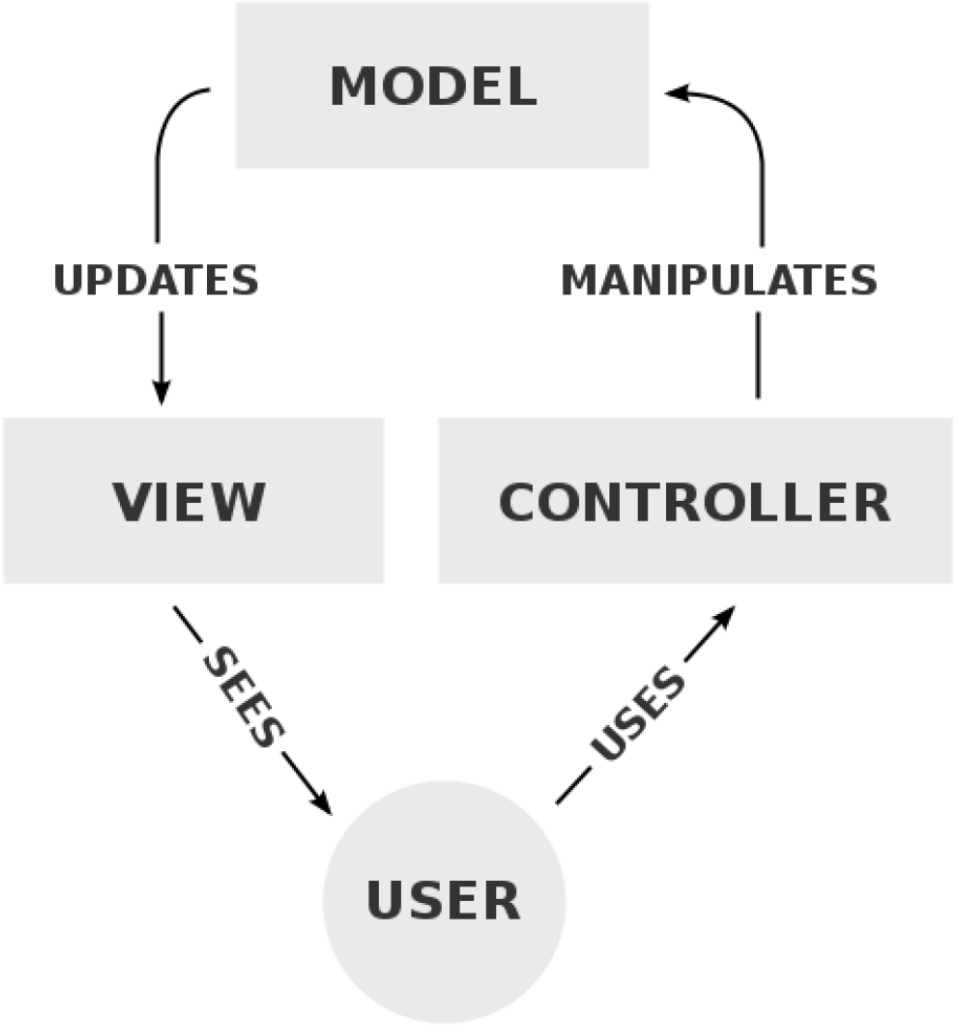

Visuals
Objects
Inputs

MVC

Design pattern for implementing interactive interfaces (like those used in VR)

Divides software into 3 components according to function of each component

Helps keep track of all the moving parts



The model, view, and controller work together to represent and change the program state

What is an application state?

Model

Manages the data, logic, and rules of the application

Primary purposes:

- Define application state
- Define how a user can change application state

Scene Graphs

Data structure for representing components of a graphics scene

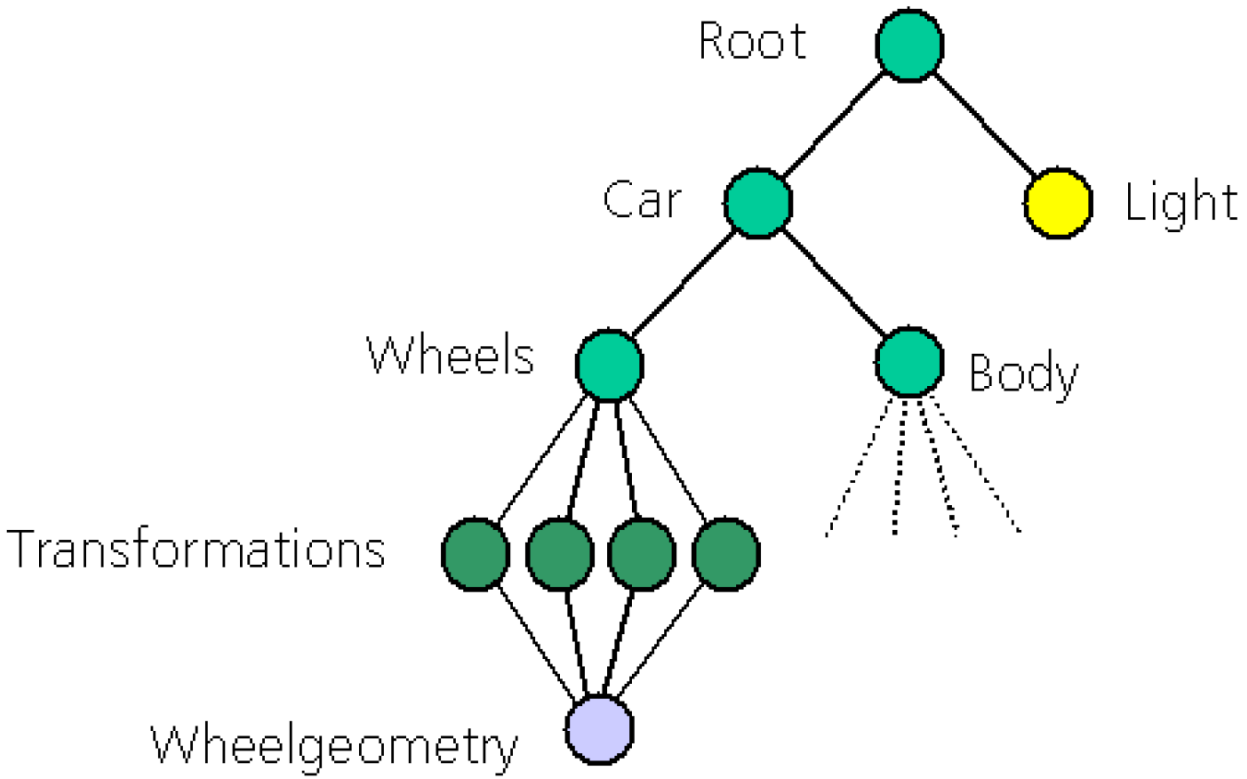
Objects are things in that scene

Scene graphs create hierarchical relationships between objects

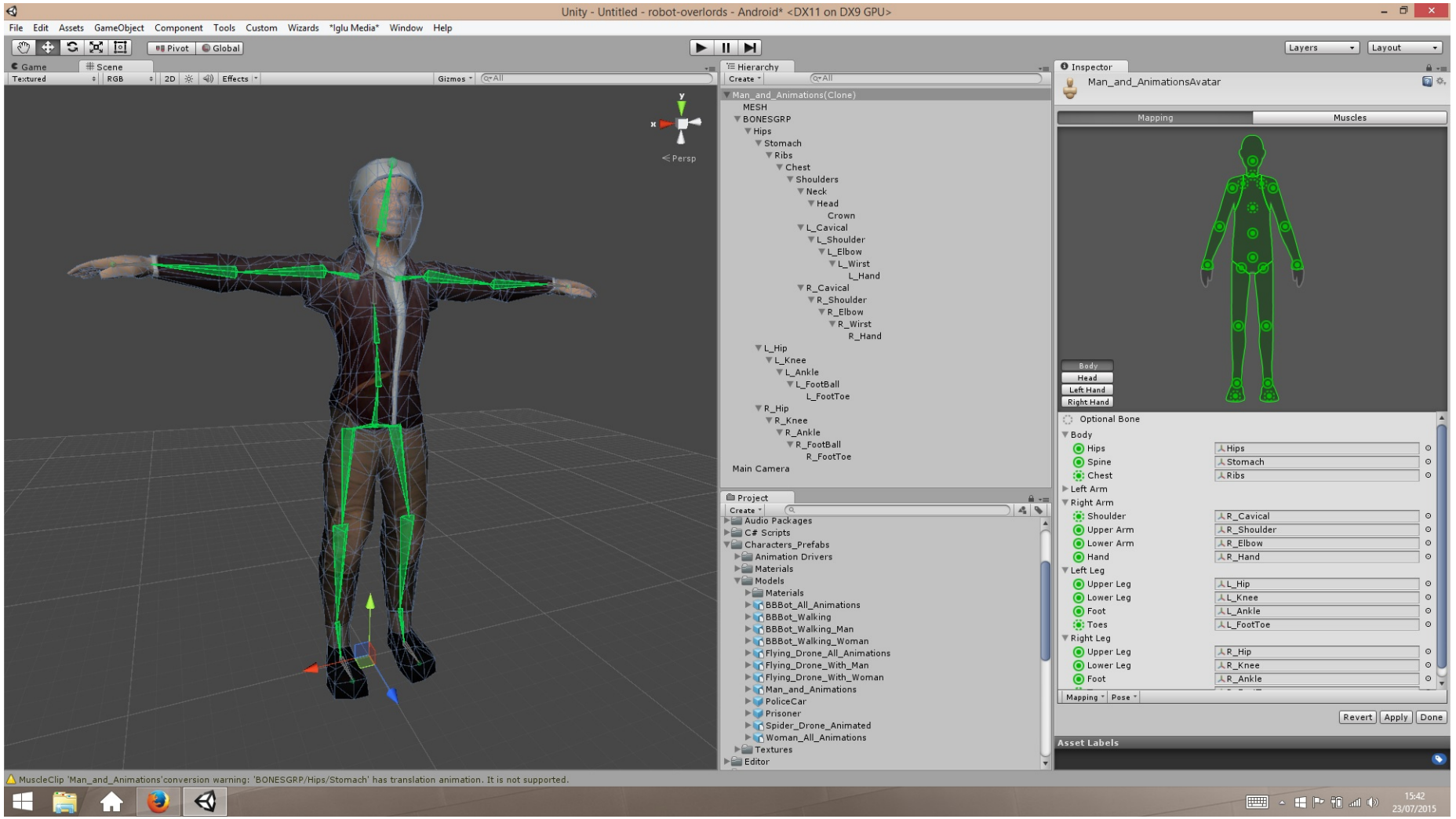
Objects themselves can also be hierarchical

An example

An example


Root
Car
Light
Wheels
Body
Transformations
Wheelgeometry

Example 2


Unity - Untitled - robot-overlords - Android* <DX11 on DX9 GPU>
File Edit Assets GameObject Component Tools Custom Wizards *Iglu Media* Window Help
Hierarchy
Man_and_Animations(Clone)
MESH
BONESGRP
Hips
Stomach
Ribs
Chest
Shoulders
Neck
Head
Crown
L_Cavical
L_Shoulder
L_Elbow
L_Wirst
L_Hand
R_Cavical
R_Shoulder
R_Elbow
R_Wirst
R_Hand
L_Hip
L_Knee
L_Ankle
L_FootBall
L_FootToe
R_Hip
R_Knee
R_Ankle
R_FootBall
R_FootToe
Main Camera
Inspector
Man_and_AnimationsAvatar
Mapping
Muscles
Optional Bone
Body
Left Arm
Right Arm
Left Leg
Right Leg
Revert
Apply
Done
Asset Labels
Project
Audio Packages
C# Scripts
Characters_Prefabs
Animation Drivers
Materials
Models
BBBot_All_Animations
BBBot_Walking
BBBot_Walking_Man
BBBot_Walking_Woman
Flying_Drone_All_Animations
Flying_Drone_With_Man
Flying_Drone_With_Woman
Man_and_Animations
PoliceCar
Prisoner
Spider_Drone_Animated
Woman_All_Animations
Textures
Editor
MuscleClip 'Man_and_Animations'conversion warning: 'BONESGRP/Hips/Stomach' has translation animation. It is not supported.

View

Generates new output to the user based on changes in the model / state

Purpose is to guide and control what the user sees

Where the renderer is housed

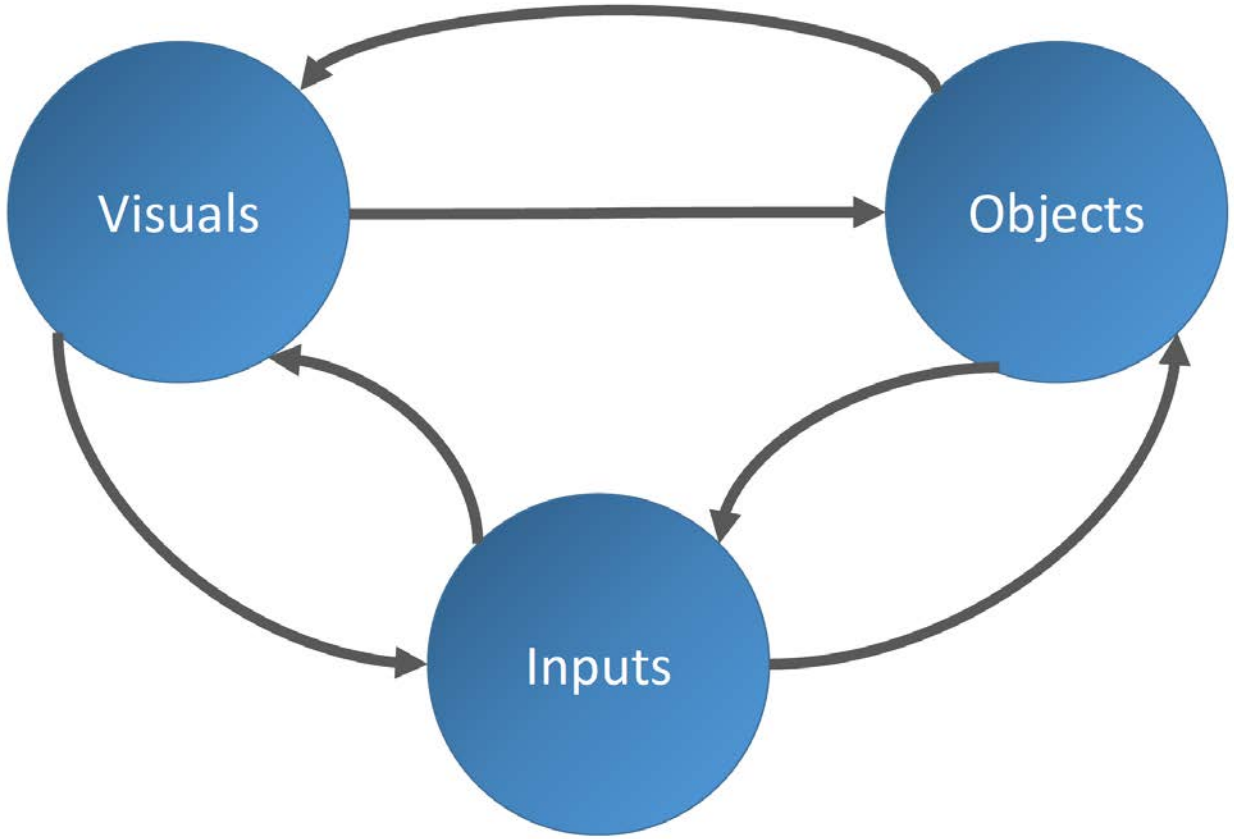
Controller

Sends commands to the model to update the model state or to the view to change the presentation of the model

Examples

- Gestures, clicks, accelerometers, gyroscope

Putting the pieces together



Views and Controllers are often tightly linked

Example: mouse input is interpreted relative to screen position and the viewer that is running at that screen position

Sometimes treat them together and call them controllers

Other times input devices, controllers, and viewers may be entirely separate

- Example: a flight simulator with a joystick

Model and controller are often weakly linked

Each controller linked to a single model

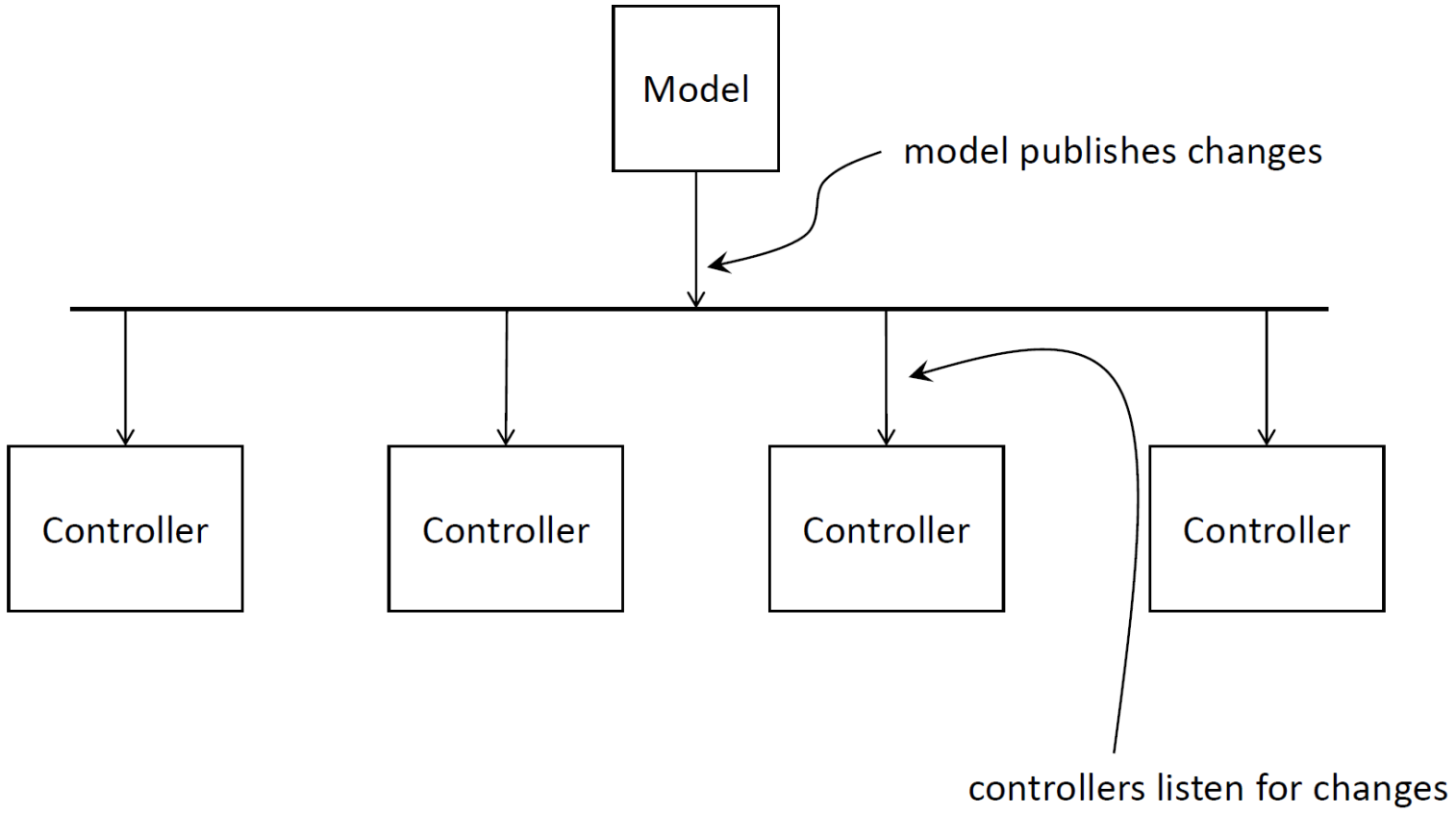
A model may be linked to many controllers

Model publishes changes in its state to the subscribed controllers

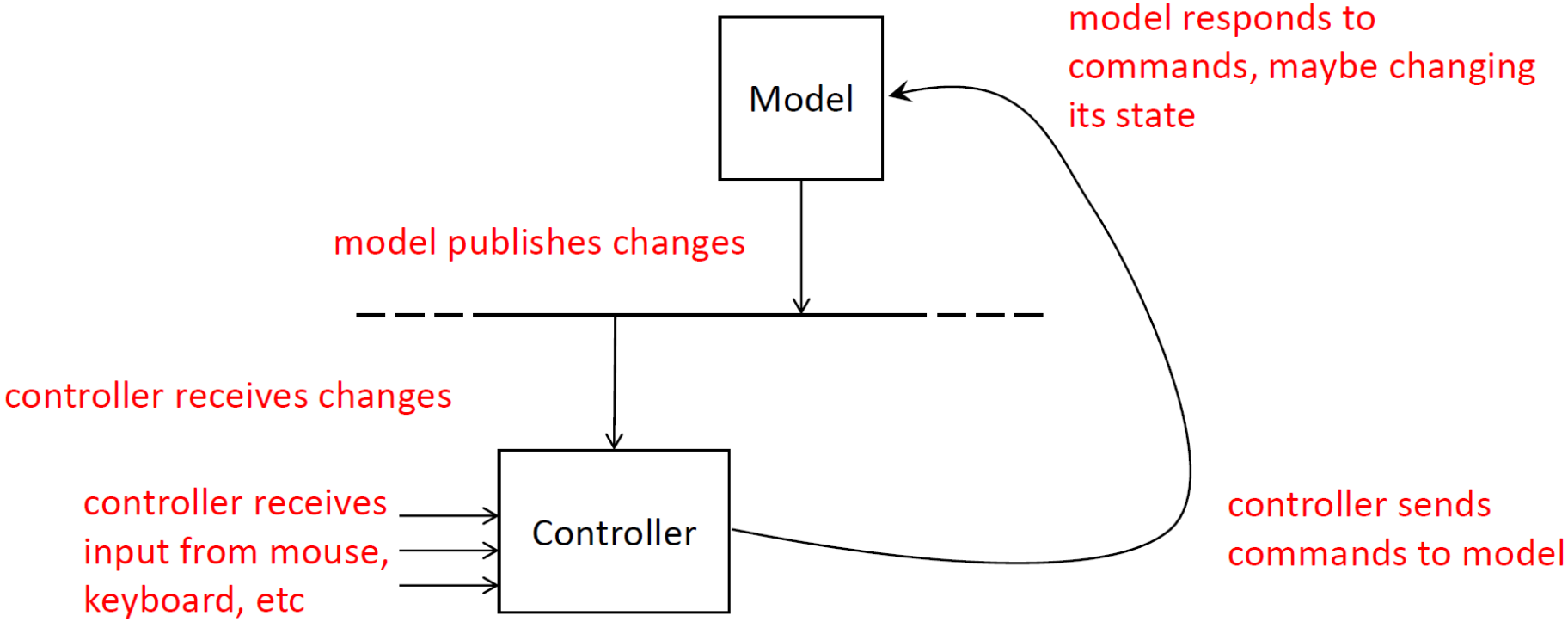
Each controller responds to the mouse and keyboard inputs by sending commands to the model

Model may change its state in response to commands it receives

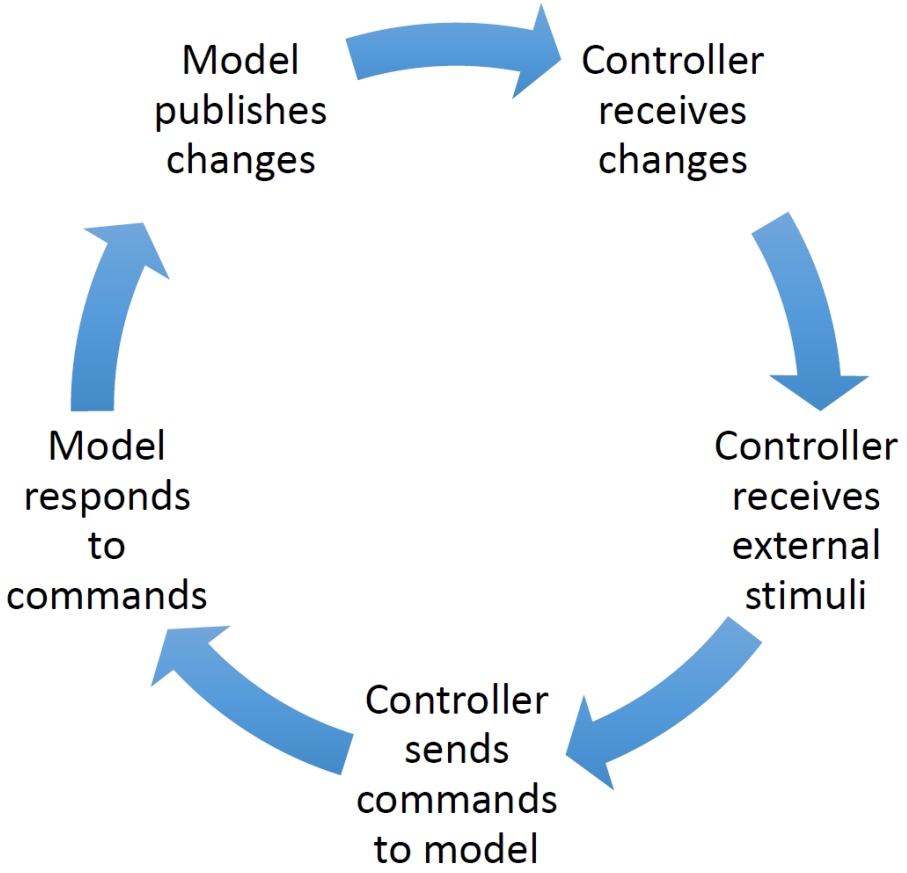
One model, many controllers



MVC feedback loop

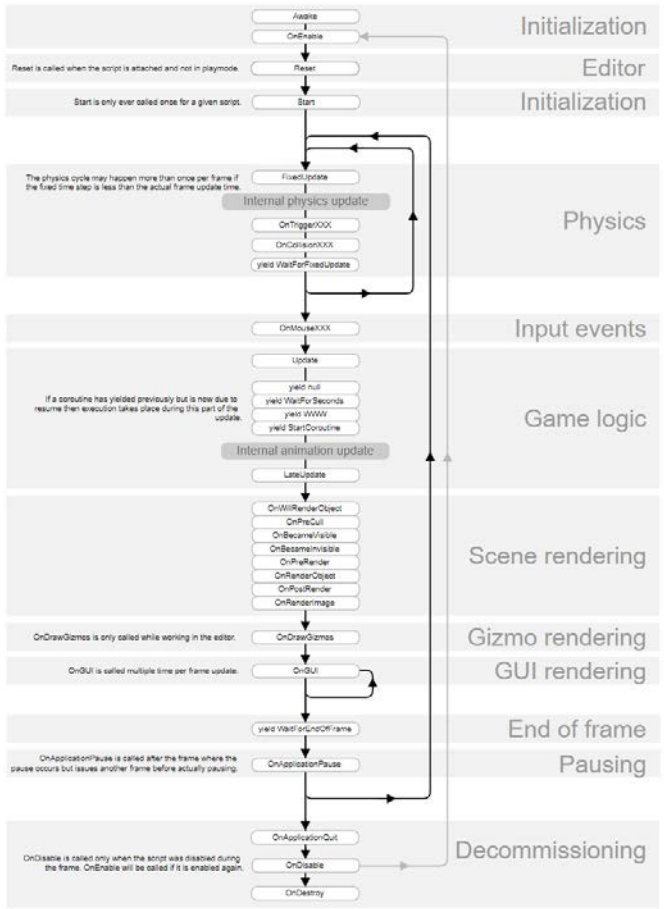


MVC feedback loop



Putting it into Practice

Application Overview



Process input

Update state

Render a frame

Application

Graphics Pipeline (or “Rendering Pipeline”) using CPU / GPU

Display

Assignment 3

We don’t have access to the internal Unity engine

Instead, we will use JavaScript build our own engine

- Assignment 3 will have you explore the ideas of MVC, hierarchical objects, and how a game engine is constructed
- Individual assignment
- Due midnight on Friday 9/21

Let’s build a simple example

University of Colorado
Boulder

THANKS!

Professor **Dan Szafir**

Computer Science & ATLAS Institute
University of Colorado Boulder