## Lecture 25:

# Toolkits for 3D Programming and the UIs of Games

05-431/631 Software Structures for User Interfaces (SSUI)

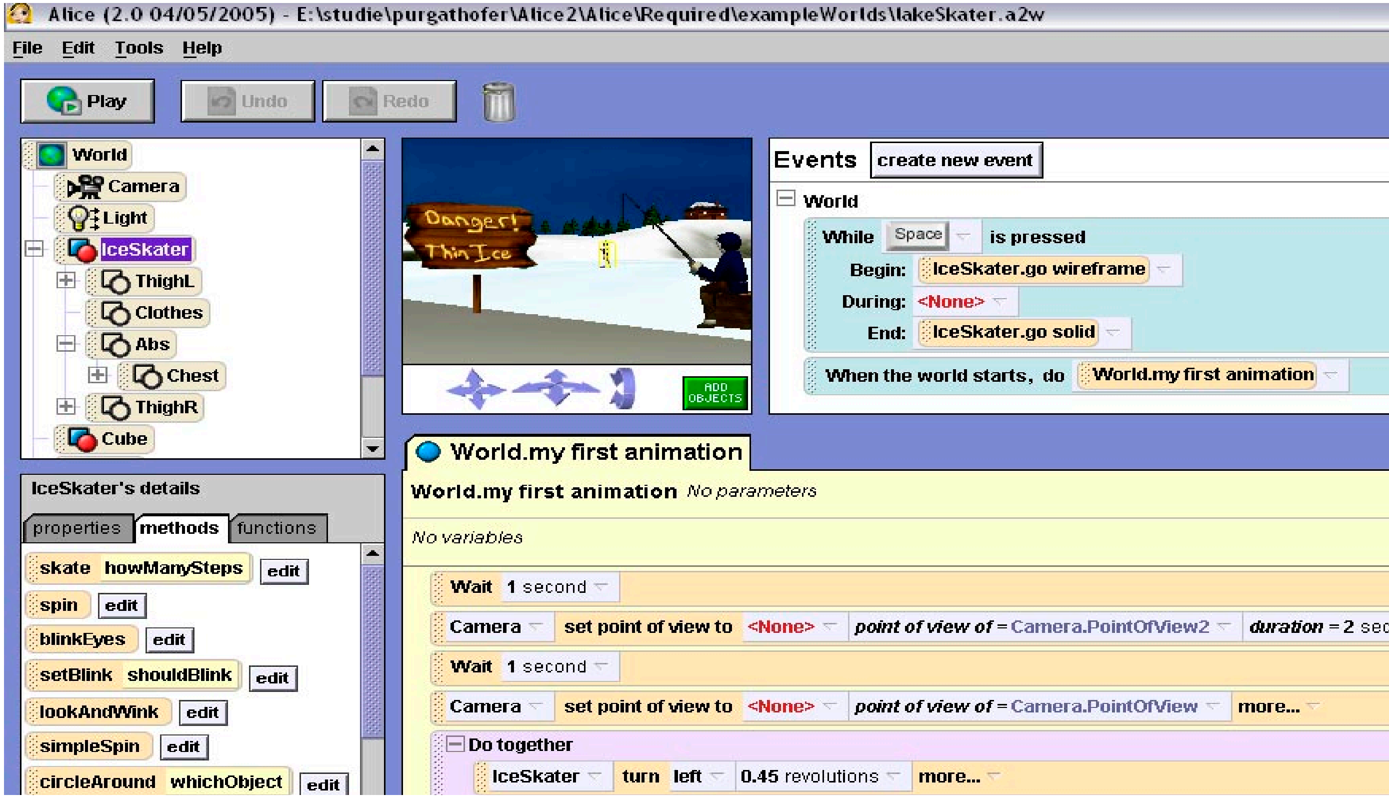
Fall, 2021

# Logistics

- Last lecture!
- Thanks for being in my class!
- Order for presentations: numerical order, OK?
  - Be sure to practice so correct length – 10 to 13 min
  - What to present and how – see https://www.cs.cmu.edu/~bam/uicourse/05631fall2021/FinalProject/index.html#presentations
  - At least a few slides & demo
- Final reports and code: due Friday, 12/10/2021 at 11:59 pm
  - Then fill out the https://www.surveymonkey.com/r/SSUI2021peer-survey
- Please ***also*** do our class survey: https://www.surveymonkey.com/r/SSUI2021Fall-Final
  - ***And*** the official CMU survey

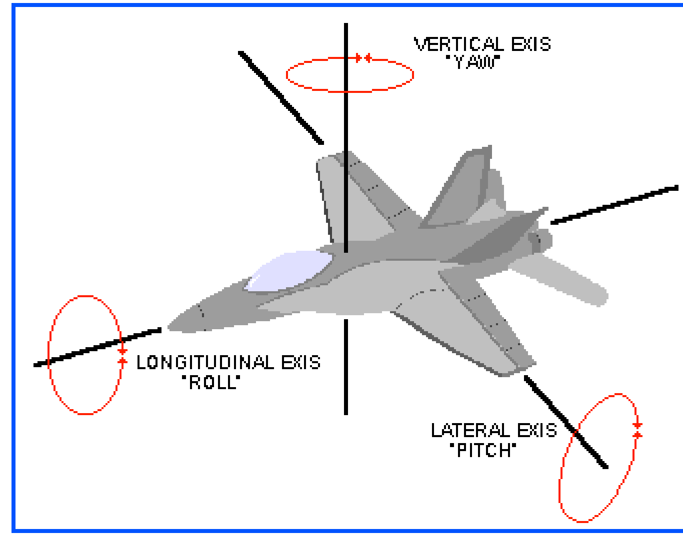
# Overview

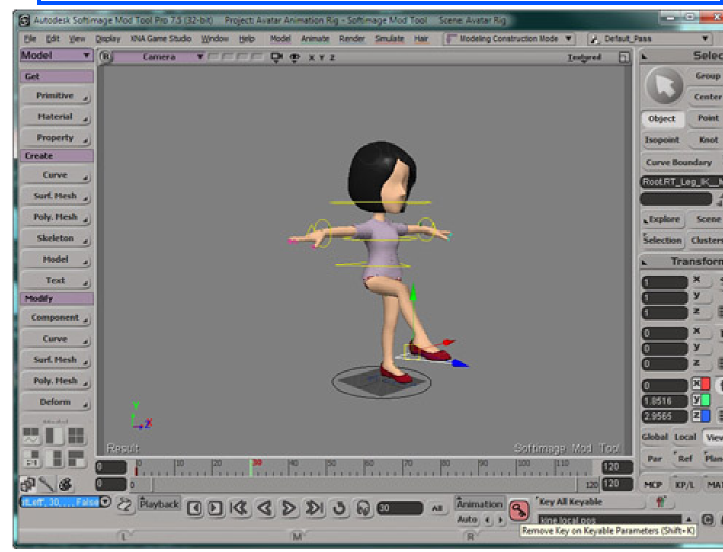
- 3D isn't just 2D +1
  - Many new issues
- Mentioned somewhat in lecture 23 on EUD tools
  - Alice 3D



# Why is 3D Harder?

- Objects have six degrees of freedom (DoF)
  - X, Y, Z
  - Roll, pitch, yaw
- Also camera position
  - Occlusion and resolution issues
  - Difficulty of orienting oneself
- People are not very good at 3D manipulation or reasoning
  - Mouse is basically 2D
- Generally, dealing with complex, hierarchical objects
- Full real-world simulation
  - Look and behaviors



# Why Hard, cont.

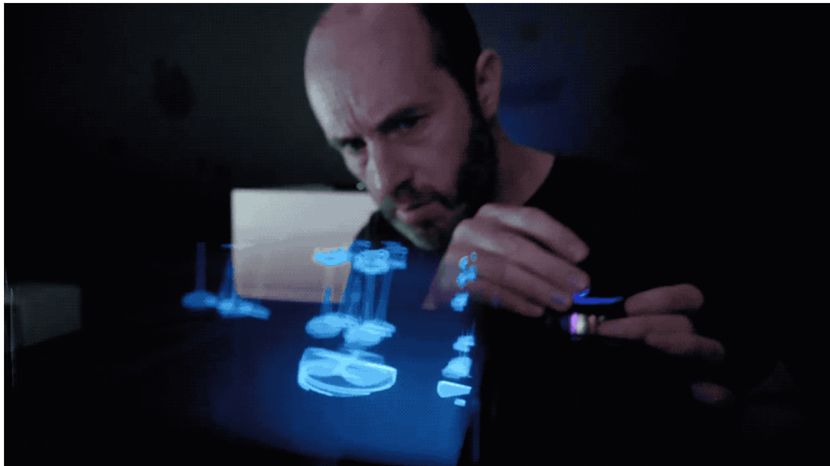
- Rick Carey, Tony Fields, Andries van Dam, Dan Venolia. 1994. Why is 3-D interaction so hard and what can we really do about it? (panel). In *Proceedings SIGGRAPH '94*. ACM, pp. 492-493. http://doi.acm.org/10.1145/192161.192299
- 3D picking is hard – which object is selected?
  - Occlusion, hierarchy, accuracy of pointing device
- Designing widgets for 3D manipulation is hard
  - Interfere with graphics
  - Should they have shadows?
- Harder to get interactive speeds for direct manipulation

# Where 3D displayed?


- Desktops – just on a screen in the usual way
- 3D "Cave" or other large displays (ACM ref)
  - Display on one or up to all walls and ceiling
- Virtual Reality (VR) or Augmented Reality (AR) headsets
  - AR – can see through the display, so pictures are superimposed on the view
- Examples:
  - Google Glass
  - Oculus (Facebook)
  - Microsoft HoloLens
  - 3D displays



Credit: https://newatlas.com/vr/voxon-photonics-3d-hologram-volumetric-displays/



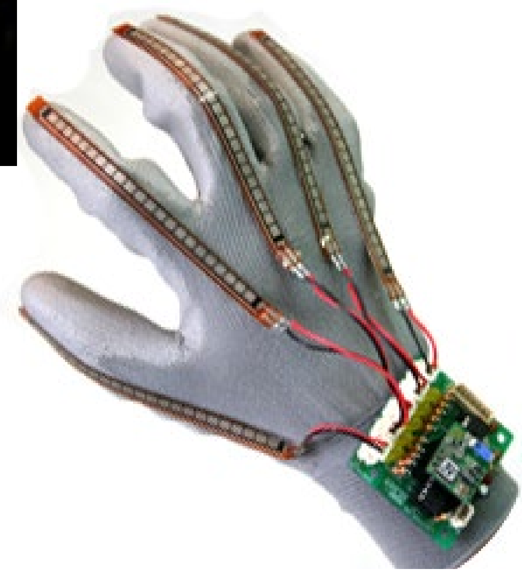
# 3D Control

- Regular Mouse or touch – 2D
  - Possibly with extra knobs or buttons
- “Mouse in the air” tracked in 3D = “bat”; 6 DoF
  - “bat” translates to *fledermaus* in German
  - (mouse that flies through the air)
- Fixed camera tracking object in 3D space
- Moving the end of an articulated motorized arm
- 3D physical objects incorporating the above

# Types of 3D sensors

- Earliest: Boxes with sets of knobs for each dimension
- Polhemus trackers (“*bat*”)
  - Starting in 1969
  - Magnetic cube on part to be tracked and nearby receiver
  - 6 DOF
  - Limited sensing area
  - Company still selling similar products
  - Often attached to gloves, head-trackers, etc.
- DataGlove
  - Starting about 1982
  - Measured finger bending = pose of hand
  - Incorporated Polhemus tracker on the wrist
- Nintendo “PowerGlove” – 1989
  - Unsuccessful – only 2 games

PATRIOT Drive Bay system with one source and two sensors.



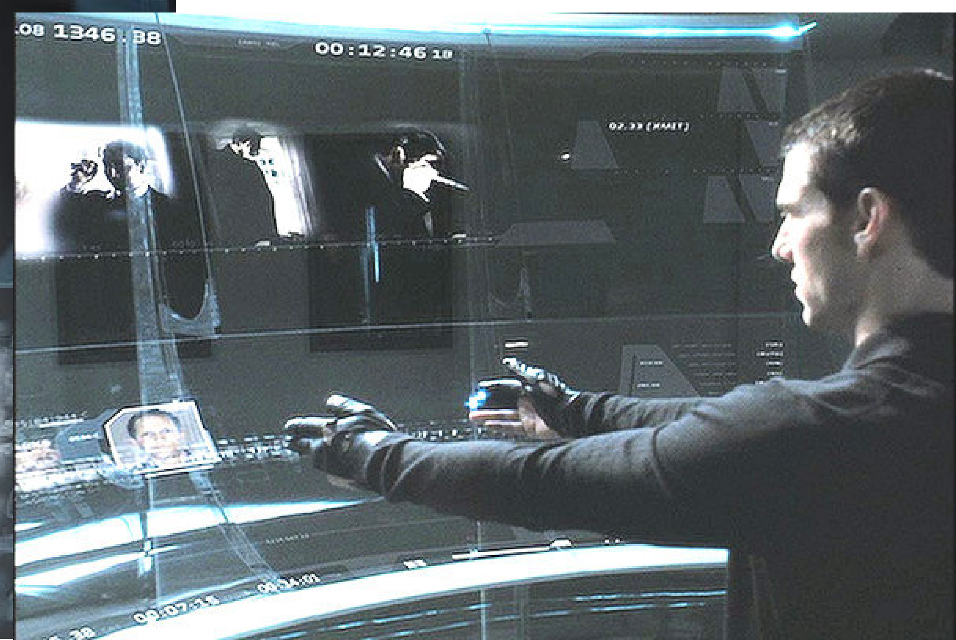
# Virtual reality on five dollars a day



- Randy Pausch. 1991. Virtual reality on five dollars a day.
  In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*, ACM, pp. 265-270.
  http://dl.acm.org/citation.cfm?doid=108844.108913
- Combined with inexpensive virtual reality headset

# Minority Report, 2002

- Using data gloves to interact with large 2-D displays in the air (or on a surface)
- MIT Media Lab advised on science (John Underkoffler)

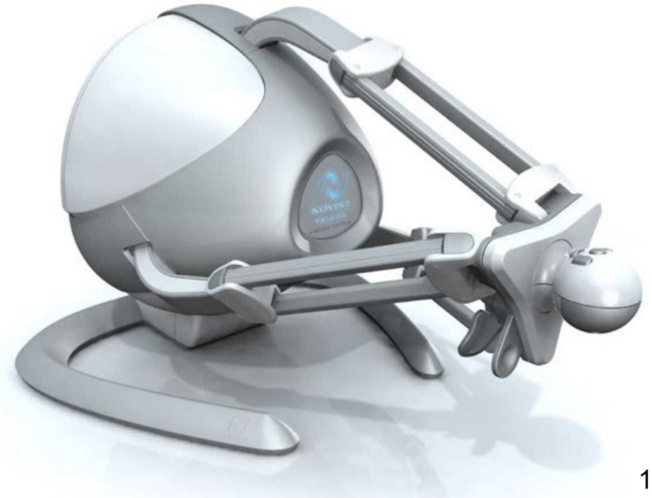


# History of 3D sensors, cont.

- Lots of motion capture research and systems
  - Motion capture rooms with cameras
  - Used for many movies, etc.
    - Example: *Alita: Battle Angel*
- Kinect
  - Introduced 2010
  - Two cameras
- Leap Motion
  - 2013
  - Camera based – designed to look upwards

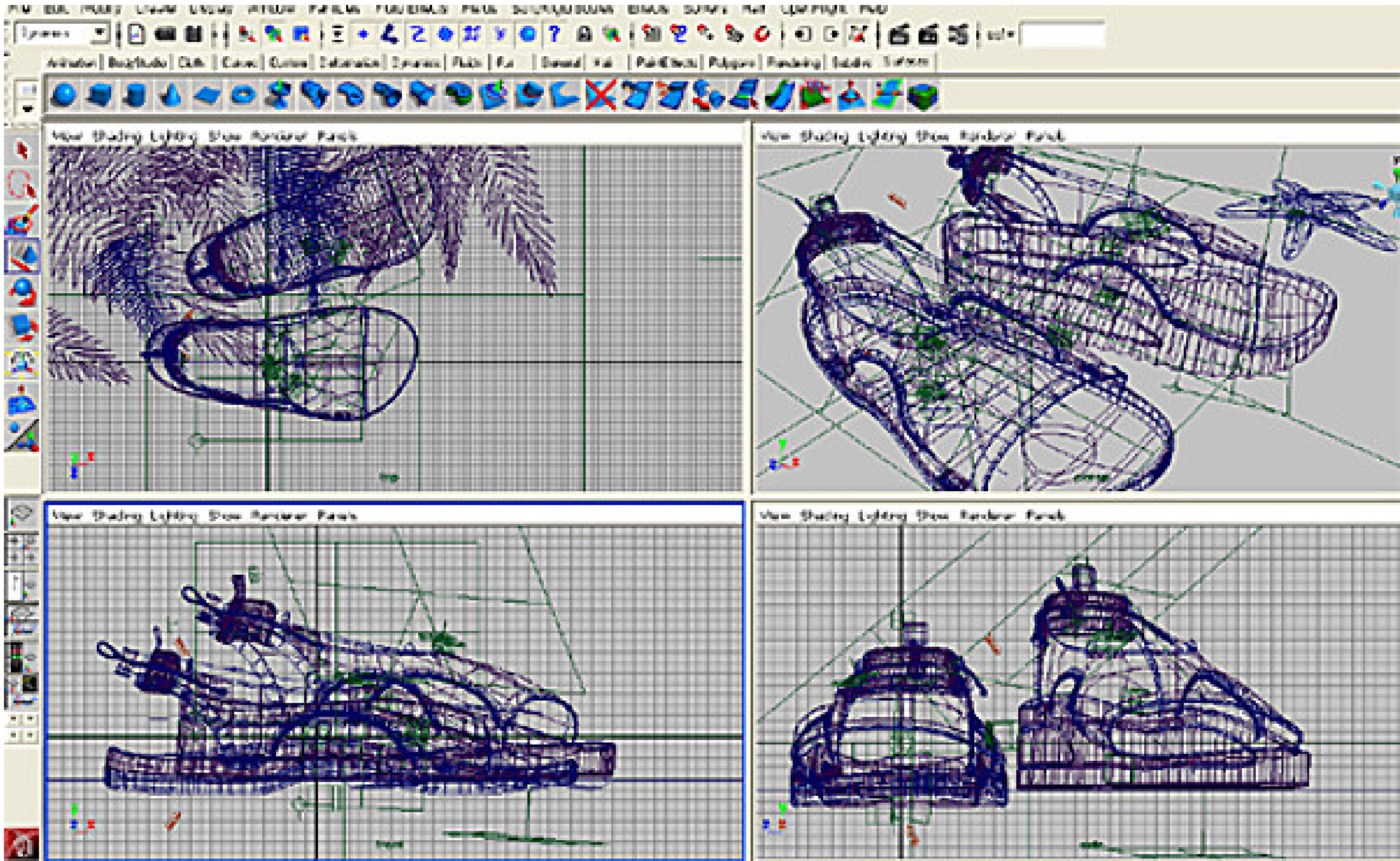

# 3D "arm" Controllers

- Motors to measure 3D movements and provide force feedback
- 3D Systems Phantom Premium
  - Medical Applications, etc.
  - 3D editing and drawing (video *0:40*)
- Falcon from HapticHouse



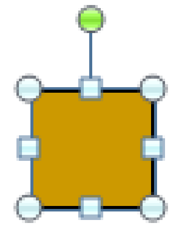
# Mouse-Based 3D manipulation

- Formerly: used 4-panel display
  - Mouse works in conventional way in each panel
  - Still tricky to manipulate
  - Now, mostly replaced with real-time motion on a single view



# 3D Handles

- Extend idea of handles on 2D objects to 3D
- Need handles for move, stretch, rotate, etc. in each dimension
  - Many approaches for doing this. E.g.,
  - Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, D. Brookshire Conner, and Andries van Dam. 1992. Using deformations to explore 3D widget design. In *Proceedings SIGGRAPH '92,* ACM, pp. 351-352. http://doi.acm.org/10.1145/133994.134091



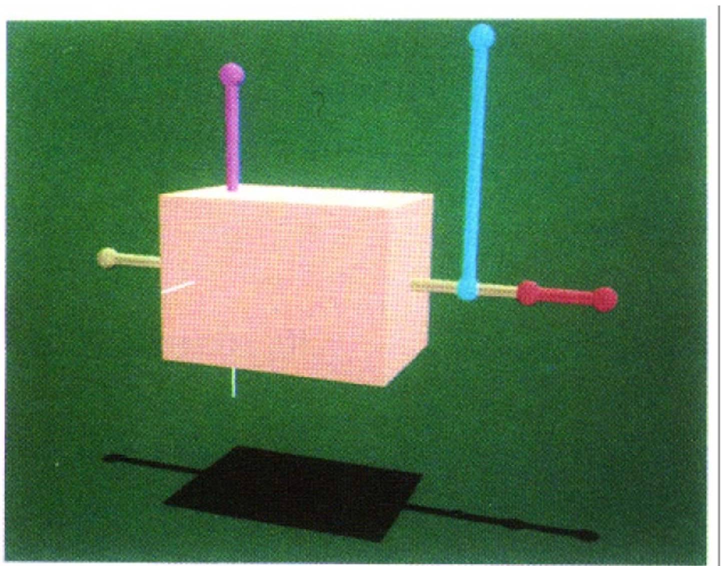


Figure 1: The starting configuration for a pink cube and a rackwith twist (purple), taper (blue), and bend (red) handles.

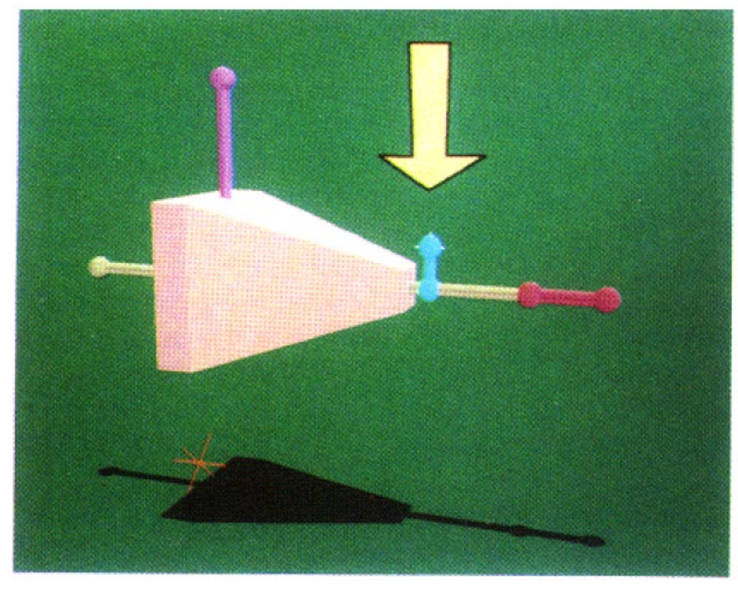


Figure 2: The taper handle is translated downward, tapering the cube. The deformation range is the region between the twist and taper handles.

# Why are games harder?

- *(Next few slides adapted from Erik Harpstead, 05-830)*

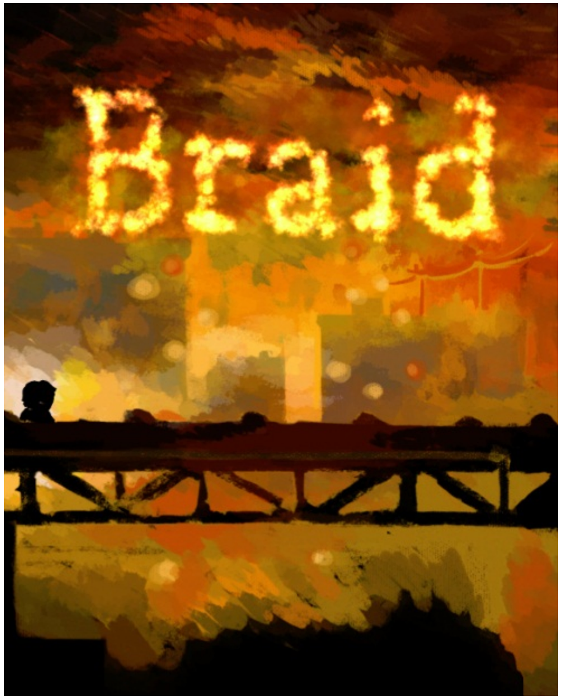
- 3D
- Rapidly shifting design requirements
- Multi-platform development
- Integration of many different forms of media (sound, music, art, modeling)
- Highly interdisciplinary teams
- The demand for novelty
- Extremely complex tools and environments

# Game Development

- Three general methods:
  - Roll your own engine
  - Use a Framework
  - Use an off-the-shelf engine

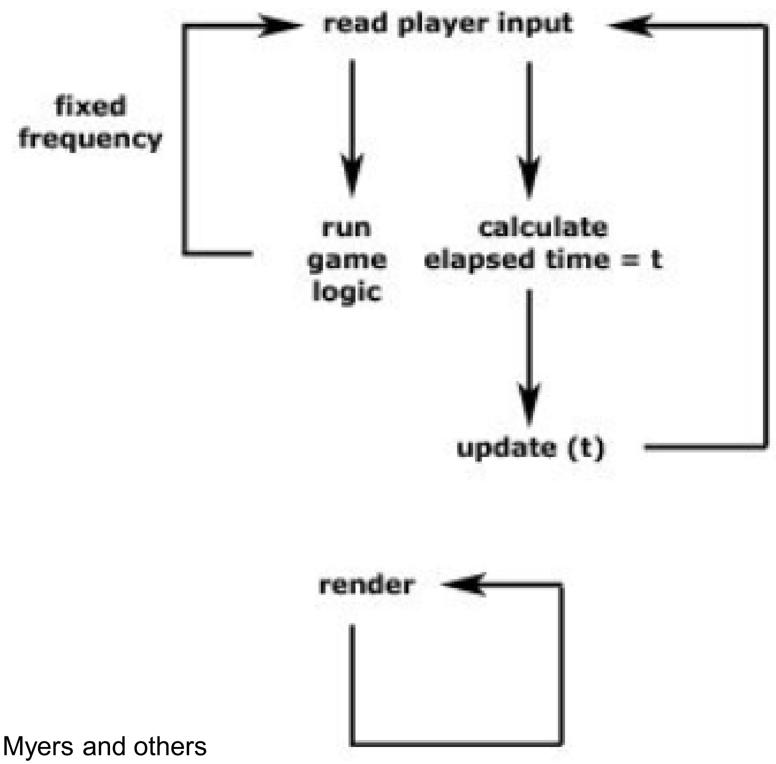
# Rolling Your Own Engine

- Surprisingly common
- Special game mechanics require custom software architectures
- Existing tools are too restrictive for rapid design changes
- Using other people's tools is a cop-out



# Using a Framework

- Usually provide basic utilities and primitives
- Commonly built around a state machine in a loop



# Using a Framework

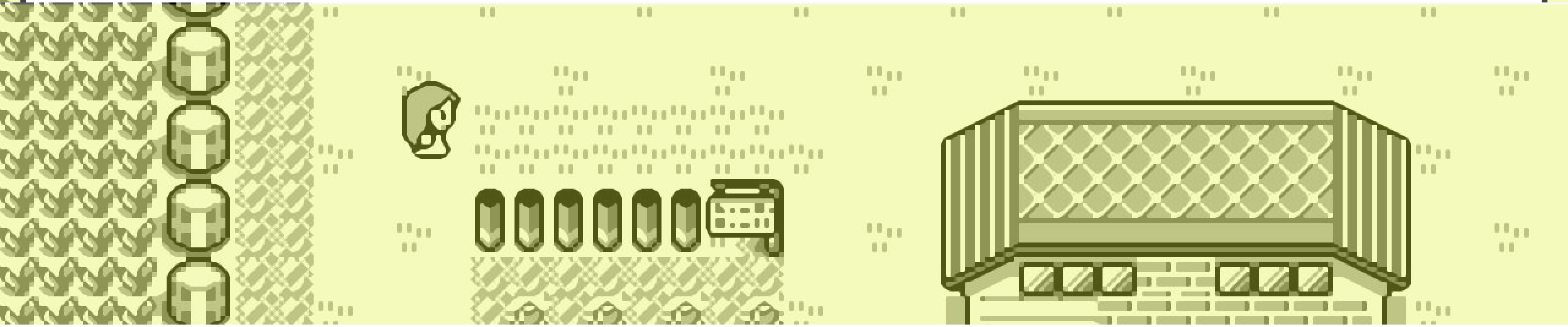
- Other Common Components:
  - Rendering Library
  - Physics Engine
  - Input Abstraction
  - Fast Math Libraries
  - Object Pooling/Resource Management
  - Audio Management

# Using a Full Game Engine

- Use some kind of interactive editor
- Provide custom API or scripting language for defining game mechanics
- More approachable by design and art members of a development team
- Combines many tools into a single package
- Examples: Unreal, Unity
- Others: (ref: https://www.incredibuild.com/blog/top-7-gaming-engines-you-should-consider-for-2020)
  - Amazon Lumberyard
  - CryENGINE
  - GameMaker: Studio (2d only, simple)
  - Godot

# Game Tools

Slides by Mary Beth Kery from 05-830 in 2017

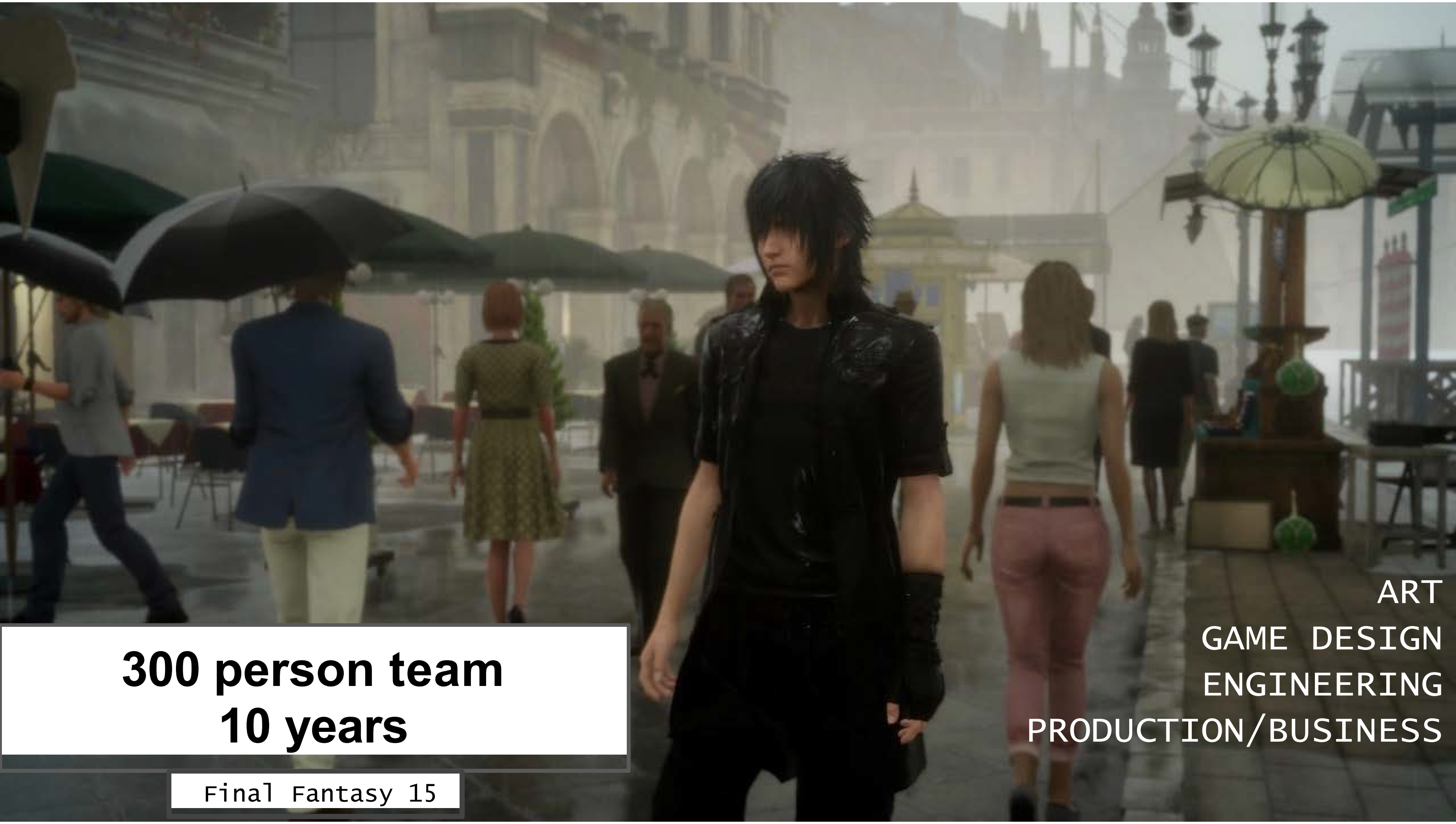


# Game Tools

MARY BETH KERY - ADVANCED USER INTERFACES SPRING 2017

# Part 1: Video game are complex software!!!


**2 person team**
**3 years**



**300 person team**
**10 years**

Final Fantasy 15

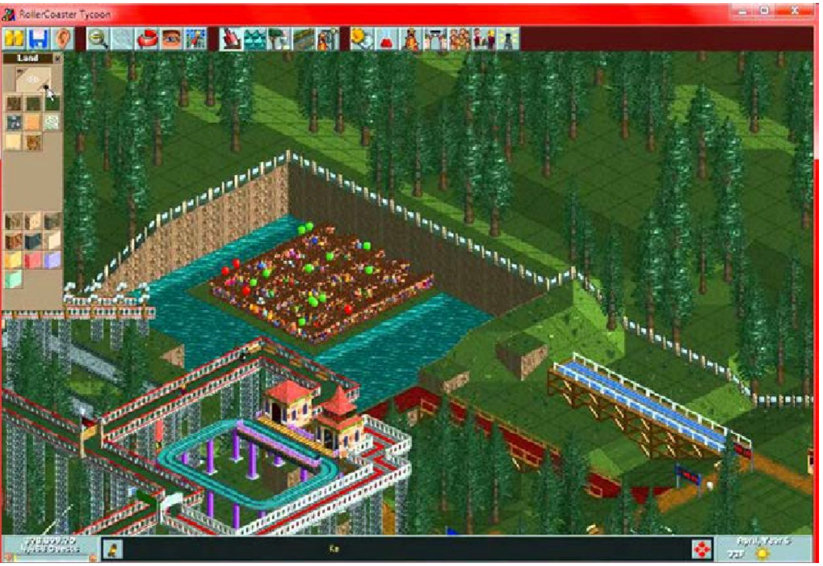
ART
GAME DESIGN
ENGINEERING
PRODUCTION/BUSINESS

# TECHNICAL CHALLENGES OF VIDEO GAMES

1. **Video games are *real time* complex simulations, and must be efficient.**

# TECHNICAL CHALLENGES OF VIDEO GAMES

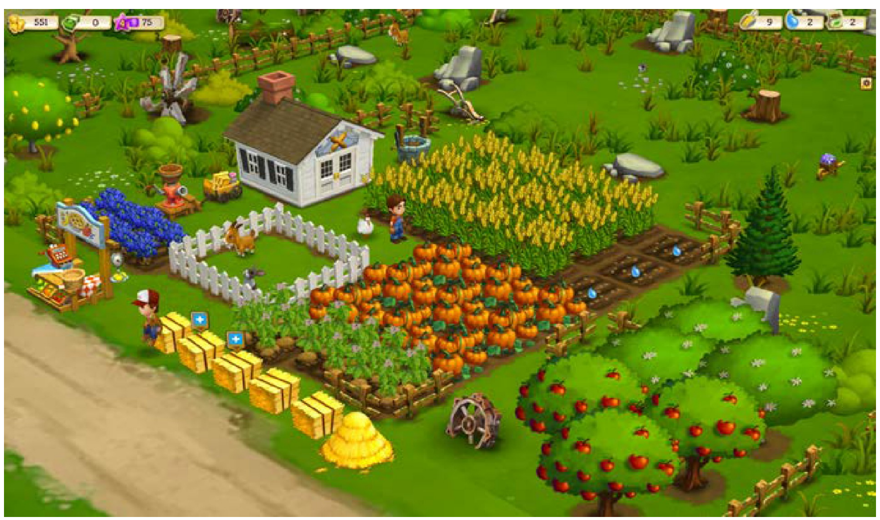
1. Video games are *real time* complex simulations, and must be **efficient**.



1999 Roller Coaster Tycoon written by one guy in x86 assembly language

# TECHNICAL CHALLENGES OF VIDEO GAMES

1. **Video games are *real time* complex simulations, and must be <u>efficient</u>.**



Today, more flexibility in language

Typically Object-Oriented

Use development tools like Visual Studio or IntelliJ

# TECHNICAL CHALLENGES OF VIDEO GAMES

## 2. People have high expectations for interactive worlds with lots of content

# TECHNICAL CHALLENGES OF VIDEO GAMES

2. People have high expectations for interactive worlds with lots of content


Lots of content on tight deadlines.

Glitches and crashes are BAD.

# TECHNICAL CHALLENGES OF VIDEO GAMES



## 3. Real time 3D graphics simulations

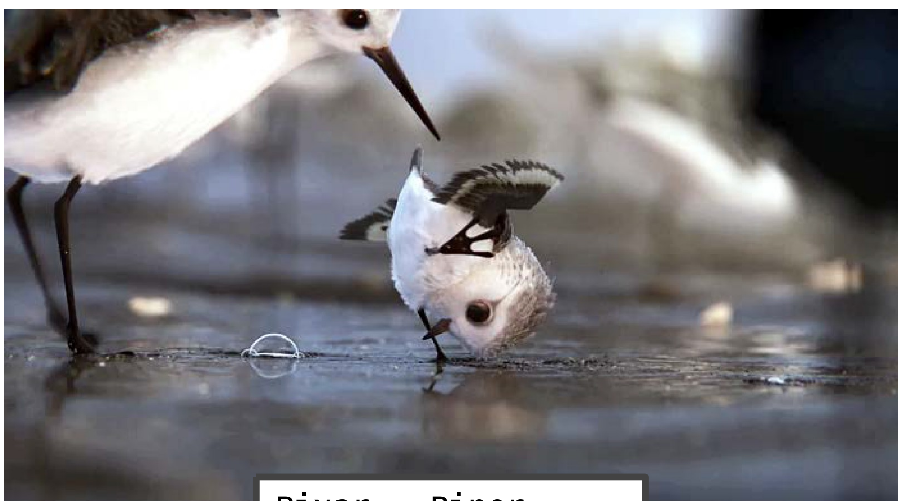

**Doom 1993**

Levels, dungeons, and rooms were not only for game pacing, but to limit the number of objects to compute and render at a time.

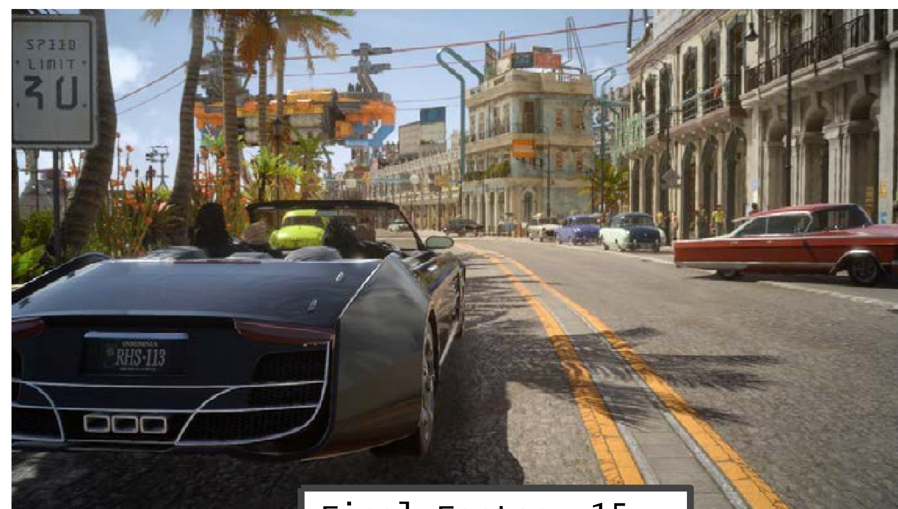
# TECHNICAL CHALLENGES OF VIDEO GAMES

## 3. Real time 3D graphics simulations

2016 graphics



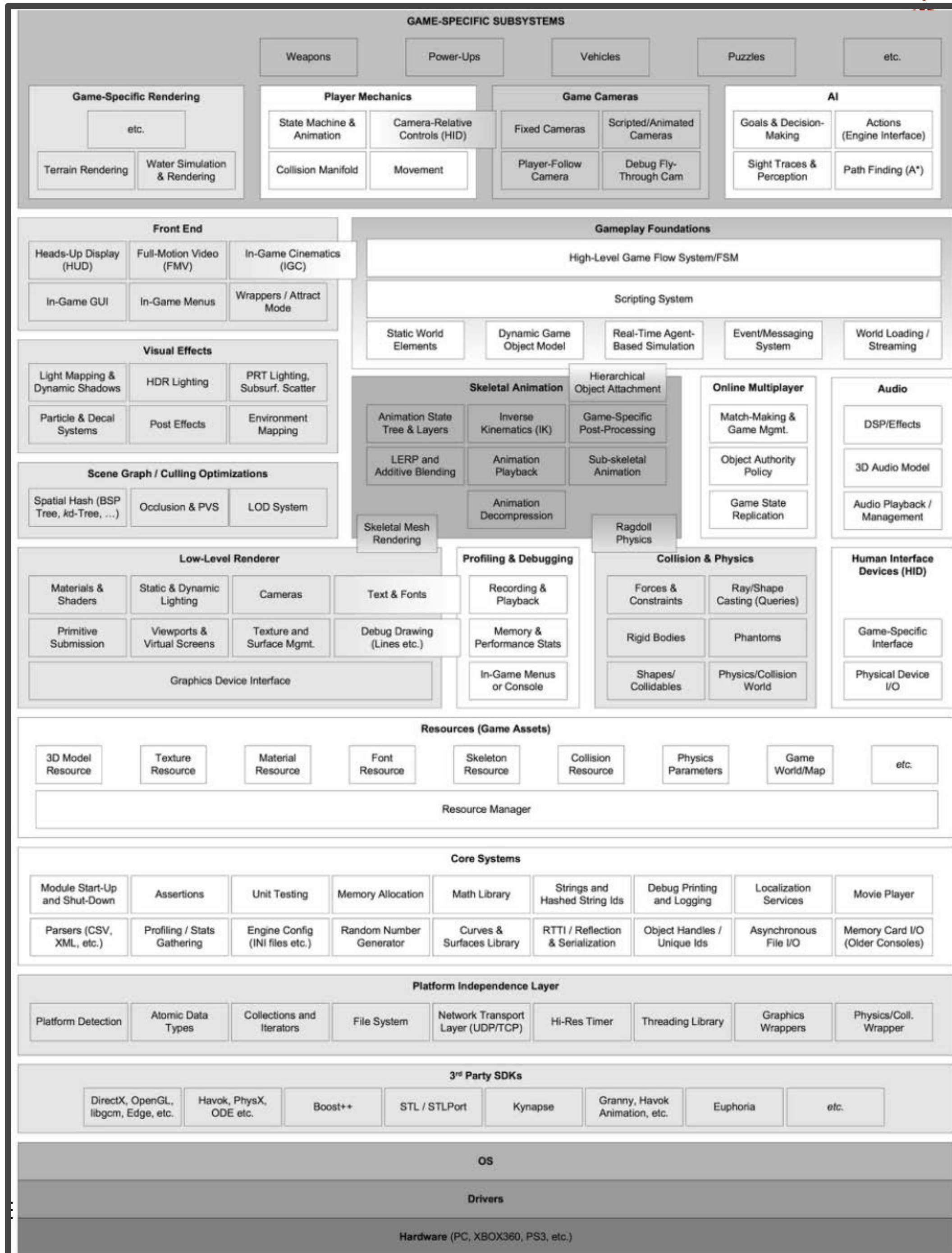
Pixar - Piper




Final Fantasy 15

# Game Engine modules

- source: Gregory, Jason. *Game engine architecture*. CRC Press, 2009.



# Game Engines: Tools that fit the pieces together

# GAME ENGINES: HISTORY

1990s First-person shooters:

**Doom by id Software**

# GAME ENGINES: HISTORY

- Architecture separates core software from game-specific assets
- ASSETS "ENGINE" SOFTWARE

**Art assets**

**Game map/ environments**

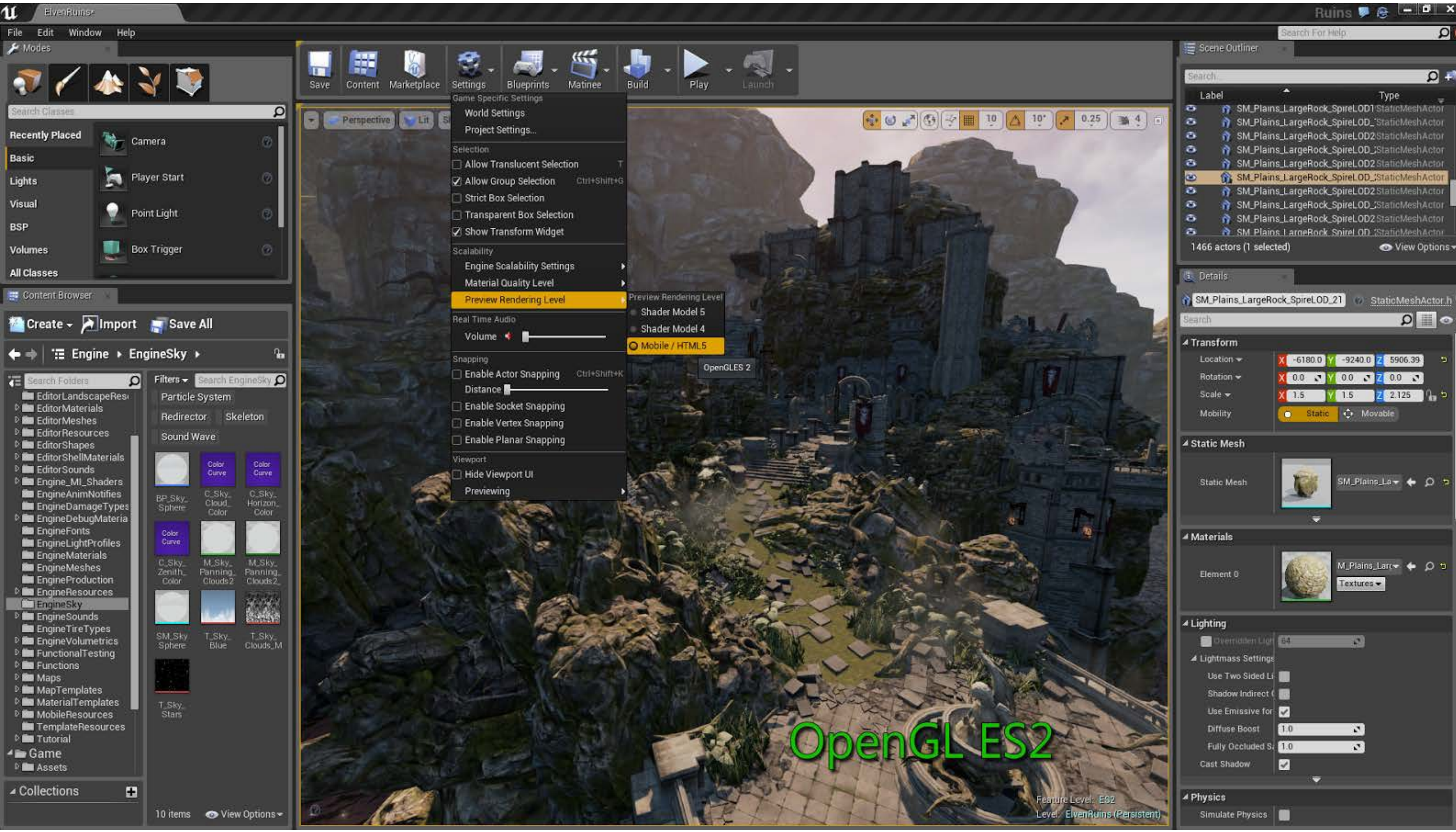
**Rules of play**

**3D graphics rendering**

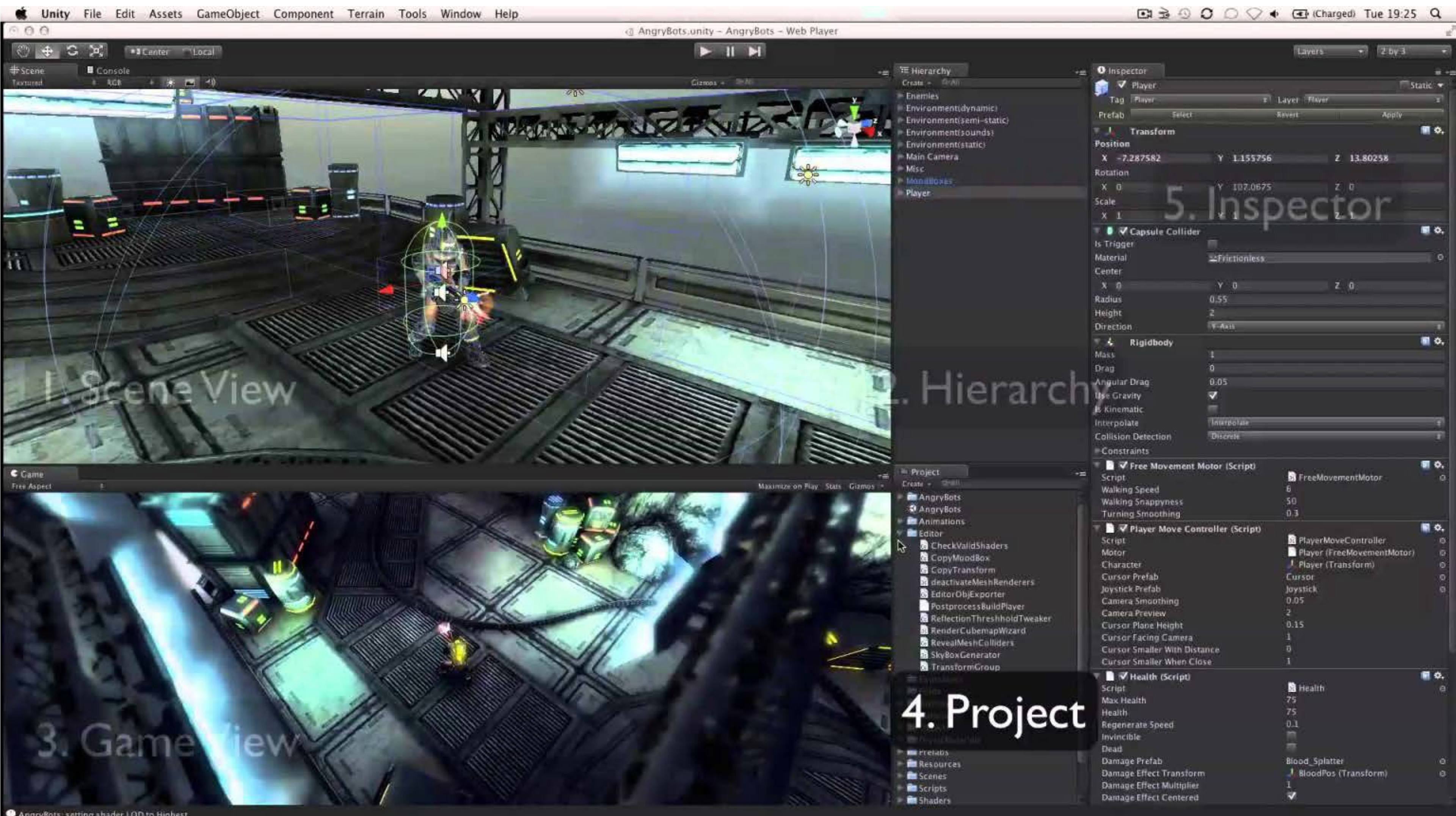
**Collision detection**

**Audio system**

# Unreal Engine: A full industry-grade development environment (advanced tool)



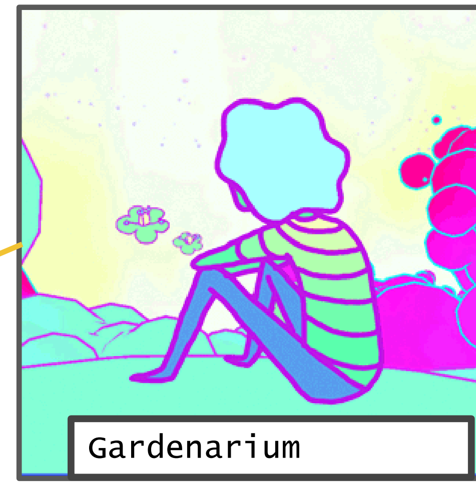
# Unity: A full development environment (advanced tool)



# A game engine has a data driven architecture that can be used to make many games

That dragon cancer

unity



Gardenarium

Clockwork

Art assets & animation

Graphics

Physics engines

Game loop

# Art assets & animation

Graphics

Physics engines

Game loop

# Art to game: Workflow of artists with tools and the game engine

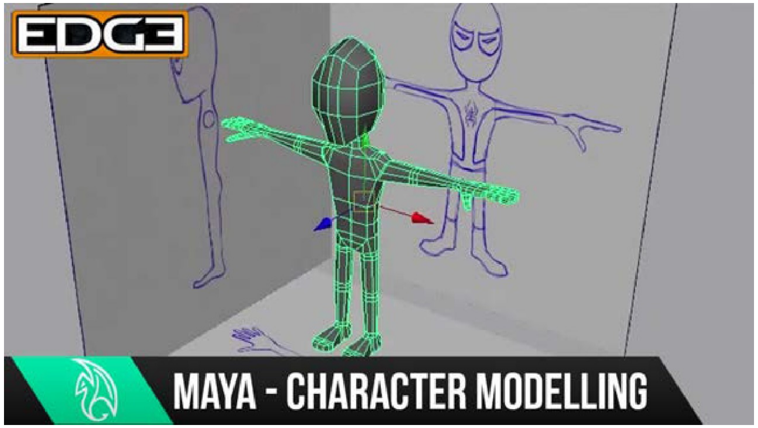

Galdin Gratin
Fresh
Boosts all stats and increases EXP earned by 10%.
HP Boost (Level 10)
Increases maximum HP by 500.
Prompto
Look out, stomach.
Next

# Photo or drawing


**The Final Fantasy 15 team cooked food and then photographed it as reference material for 3D modelers and shaders.**

# 3D Scanning or image tracing

The Final Fantasy 15 team scanned their food and photographed it




Modelers use reference drawings from different angles

# Modeling Software

AUTODESK® MAYA®

blender™

model in progress

Final in-game model.

# Textures and Shading

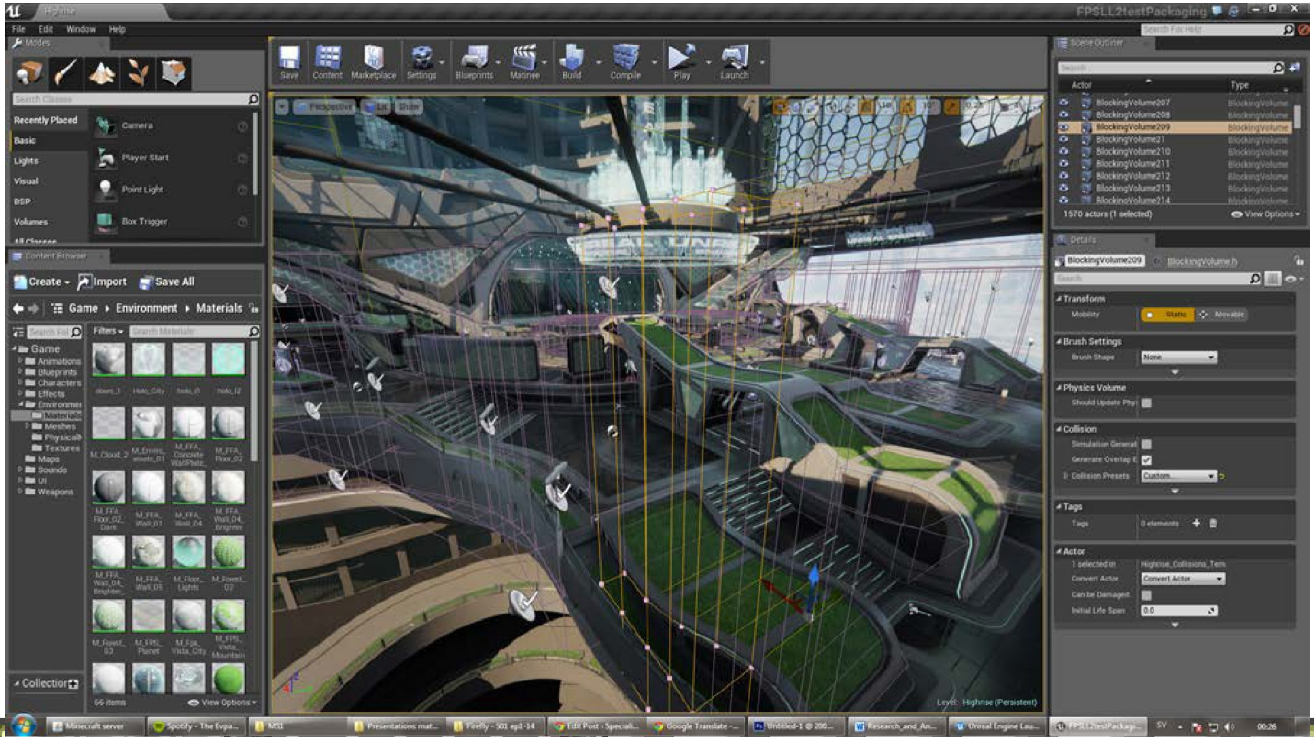
AUTODESK® MAYA®

blender™

Final in-game model.

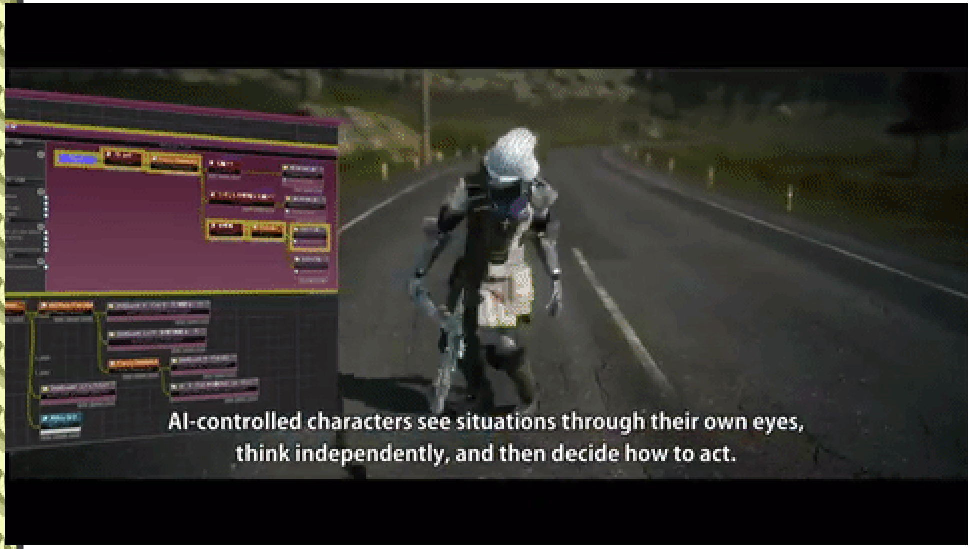
# Back to the game

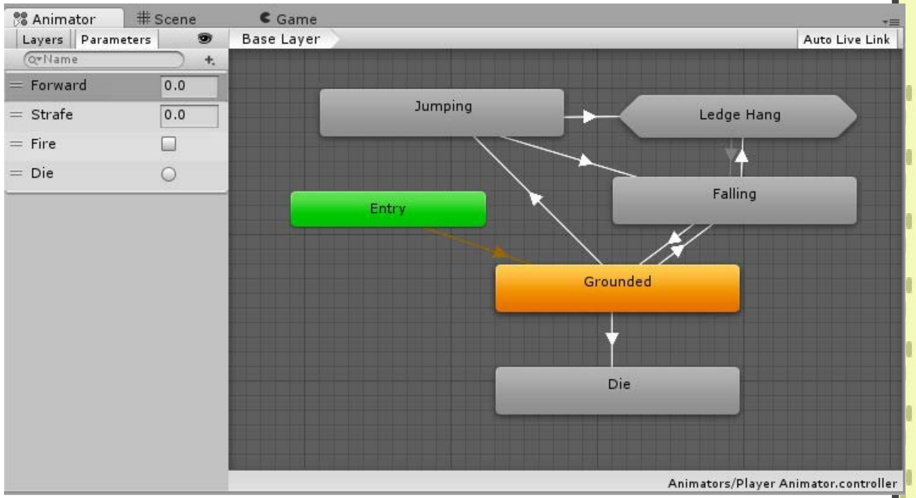
**Unreal Engine place objects in scene with map editor**



# In the game engine

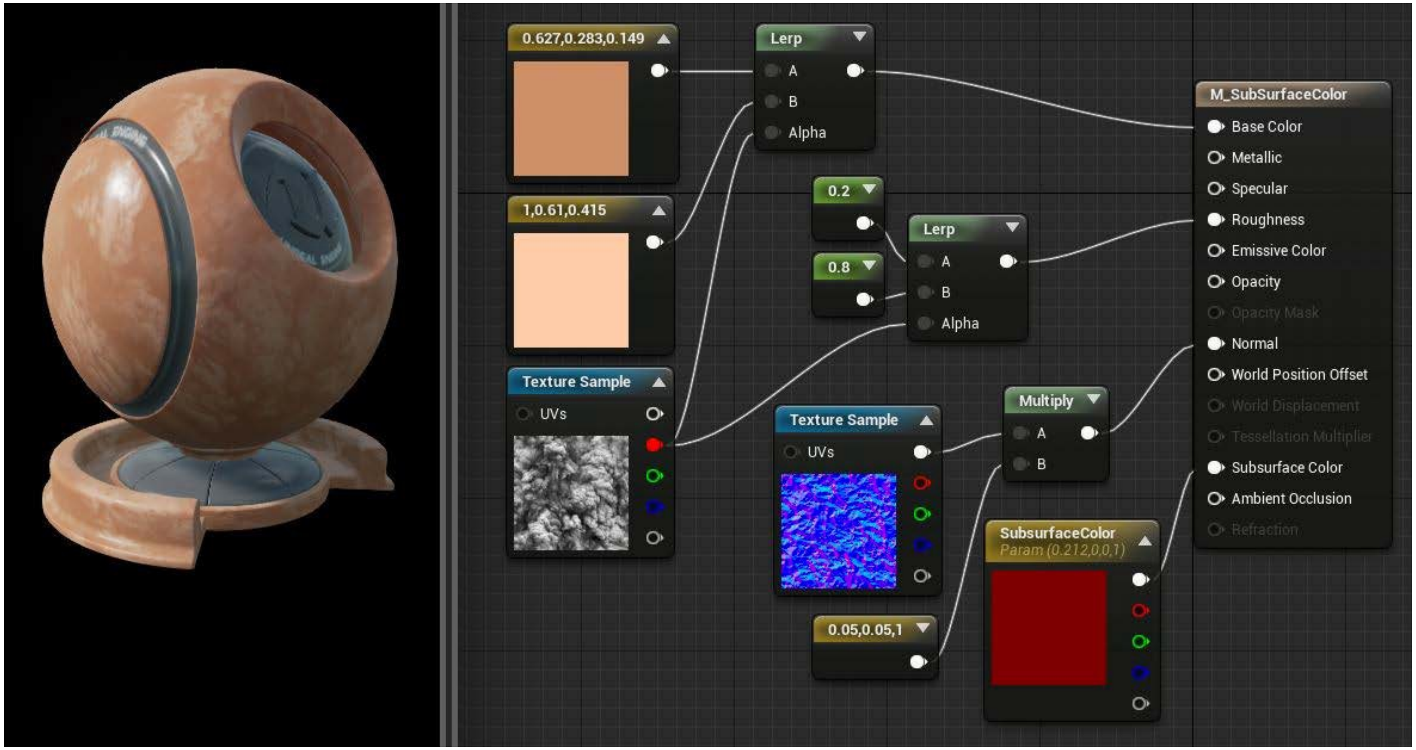
**Visual programming languages allow animations, materials, and shaders to be written by artists**



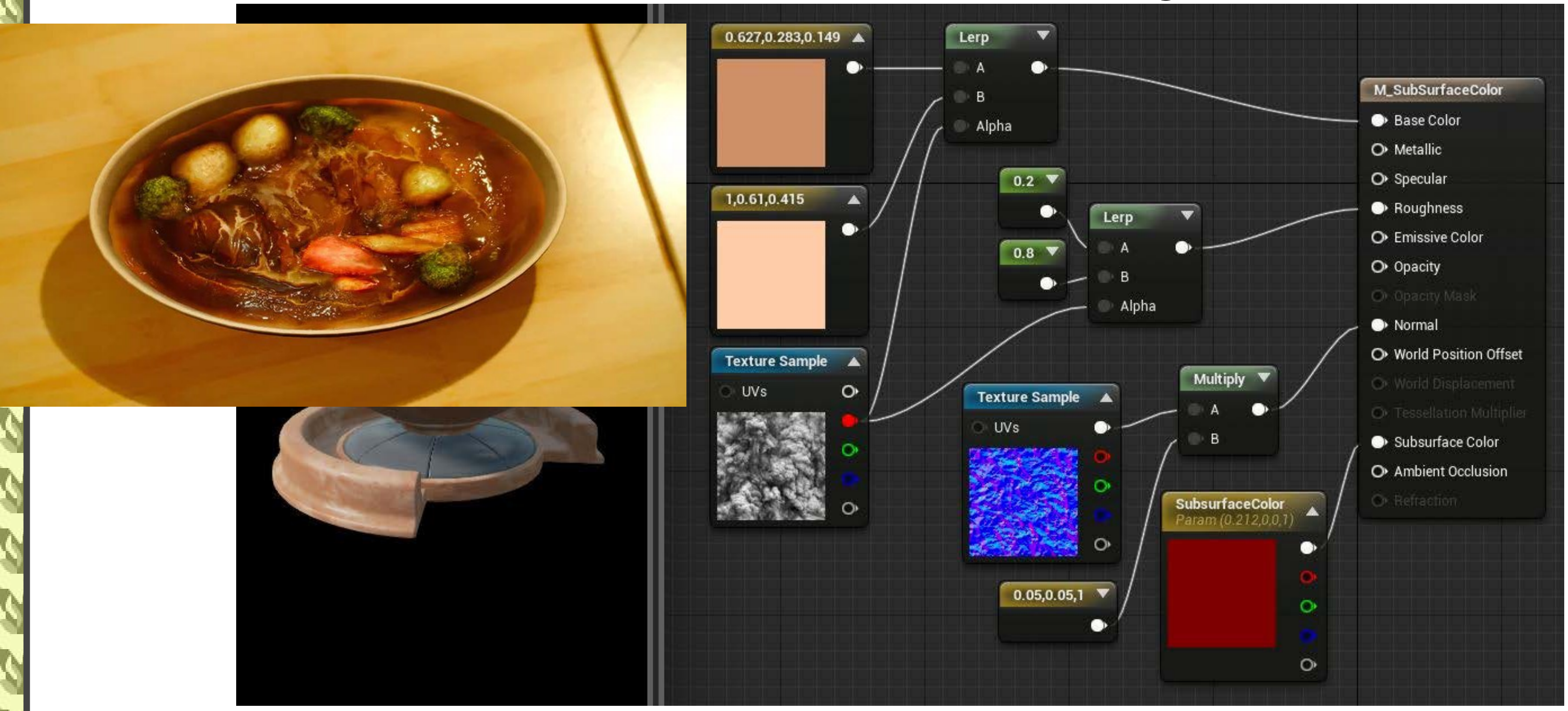
# In the game engine

**Visual programming languages allow animations, materials, and shaders to be written by artists**



# In the game engine

**Visual programming languages allow animations, materials, and shaders to be written by artists**



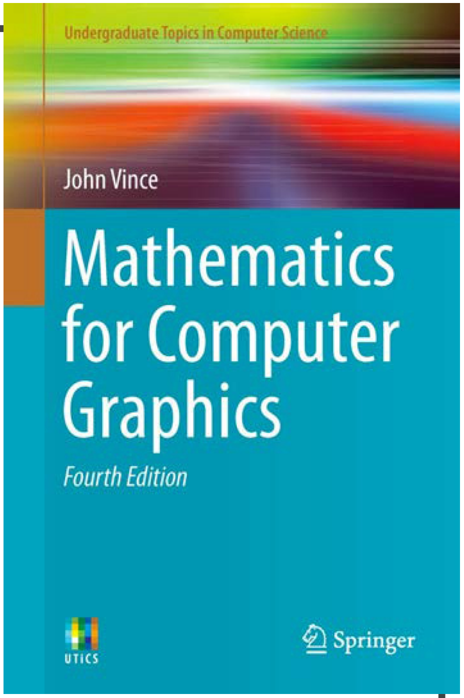
Art assets & animation

**Graphics**

Physics engines

Game loop

# Shaders = VERY TECHNICAL




COMPUTER GRAPHICS! 🎉

# Technical Graphics Tools

OpenGL®

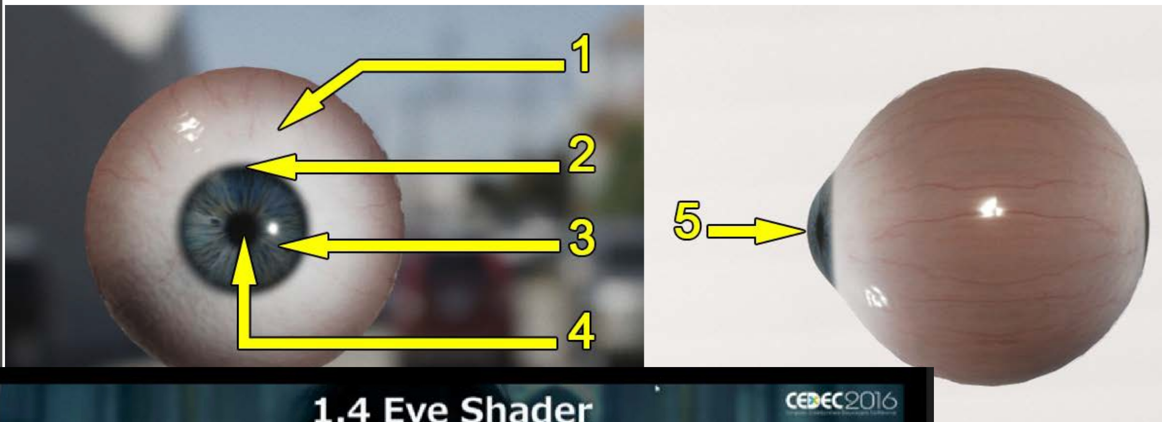
**Open GL has bindings in lots of different languages**

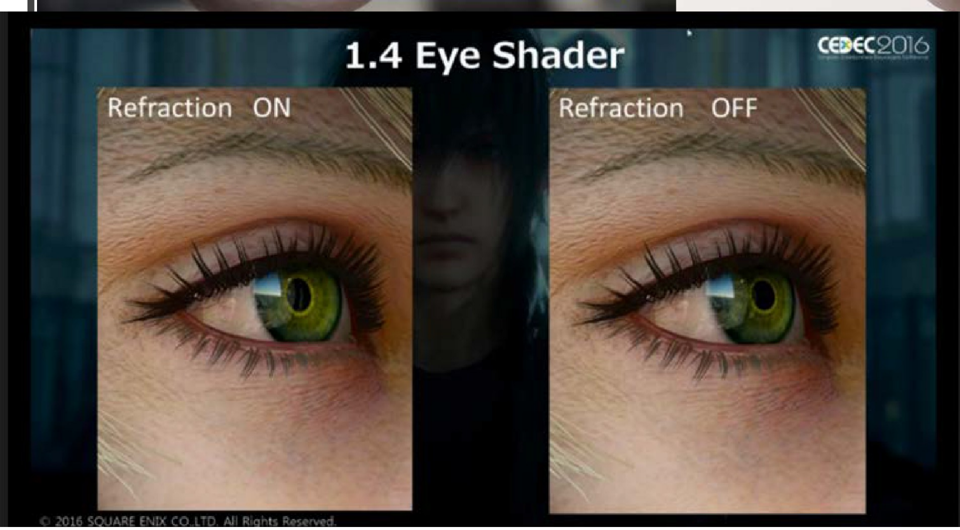
**Powerful, but not easy to learn.**

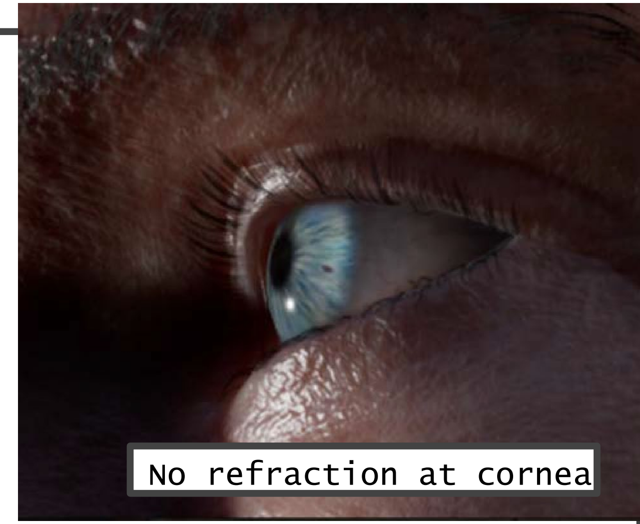
three.js

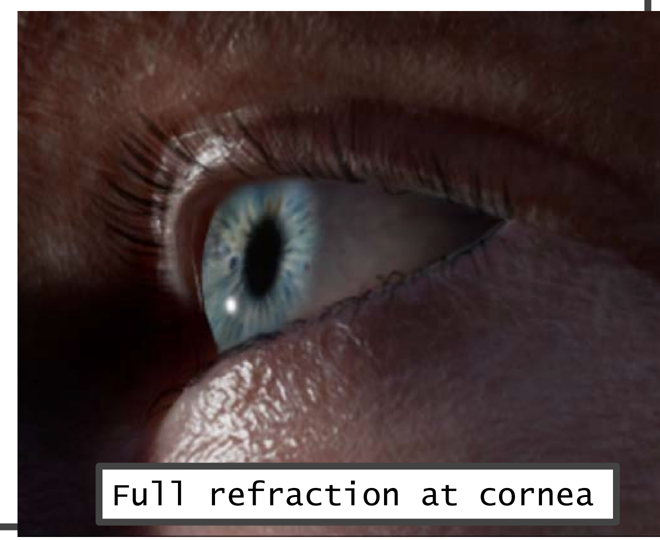
**Some language bindings are more learner-friendly than others**

# Technical Graphics – eyes



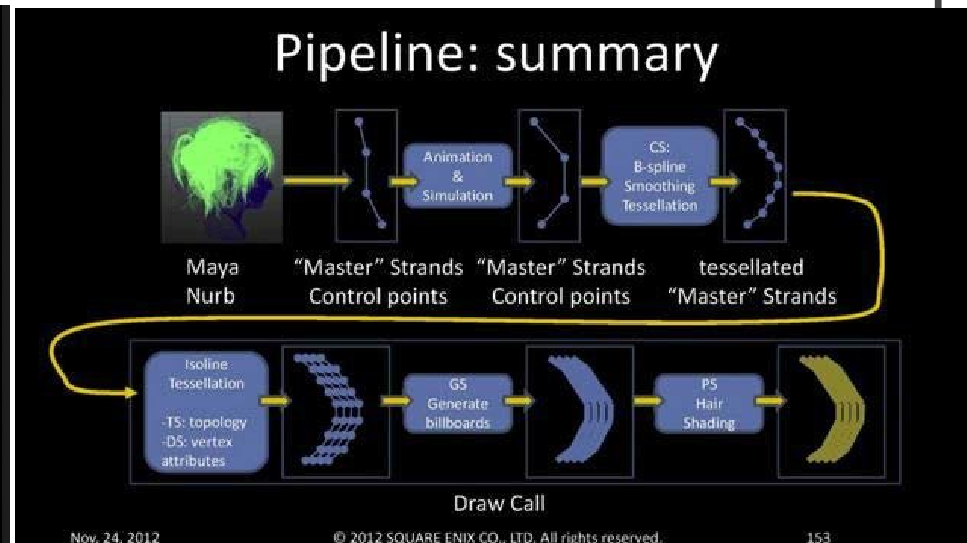



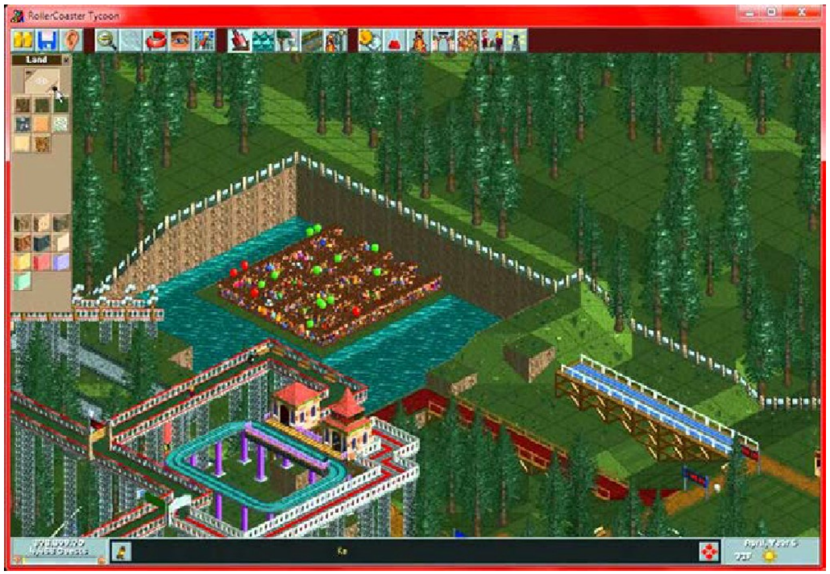
# Technical Graphics – hair

Human-Computer Interaction Institute


**Process of modeling and rendering character Lunafreya's hair from Final Fantasy 15x**

# Graphics – Updating the Screen




## Must be efficient!

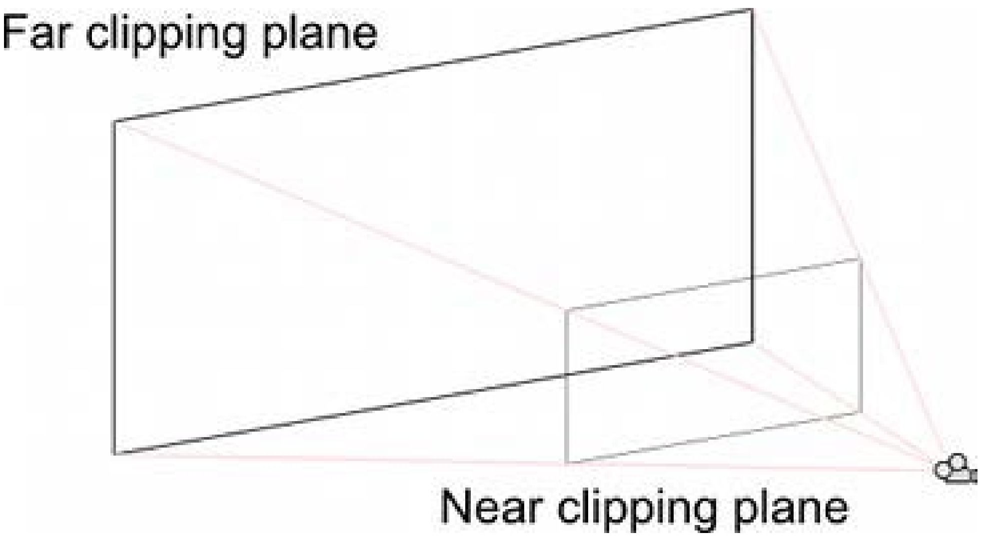
**The screen must be updated every frame, at 30fps to 60fps. Rendering and shaders are computationally expensive!**

# Graphics – Updating the Screen

**Occlusion culling problem: don't render hidden objects**

**Frustum culling: test if an object intersects with the frustum.**

**Portals: designers *manually* place simple primitives around chunks of the game world. The portals are invisible but cheap to test intersection on.**

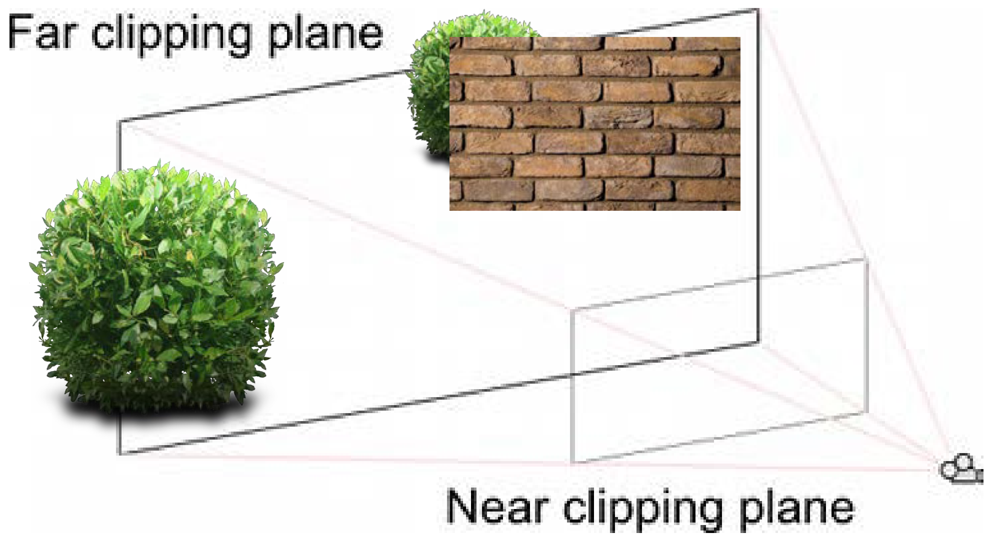


# Graphics – Updating the Screen

**Occlusion culling problem: don't render hidden objects**

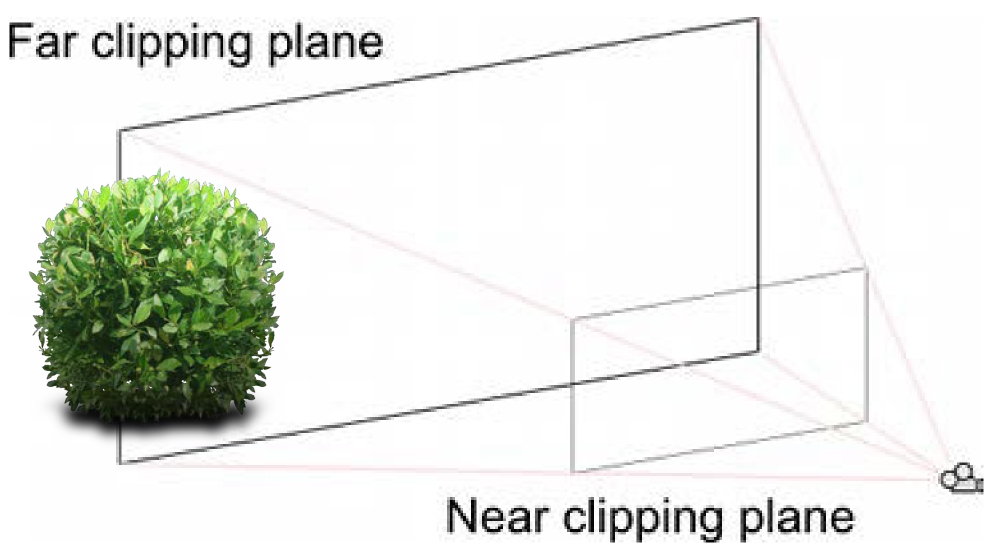
**Frustum culling: test if an object intersects with the frustum.**

**Portals: designers *manually* place simple primitives around chunks of the game world. The portals are invisible but cheap to test intersection on.**



# Graphics – Updating the Screen

**PVS: Potential Visibility Set, precomputed. Very efficient for small environments. PVS is submitted to the renderer and items in the set are tested to make sure they are indeed visible**
**Bad: storage costs**



Art assets & animation

Graphics

**Physics engines**

Game loop

# Physics

**Unity or Unreal game engines have basic built-in libraries.**

## Physics

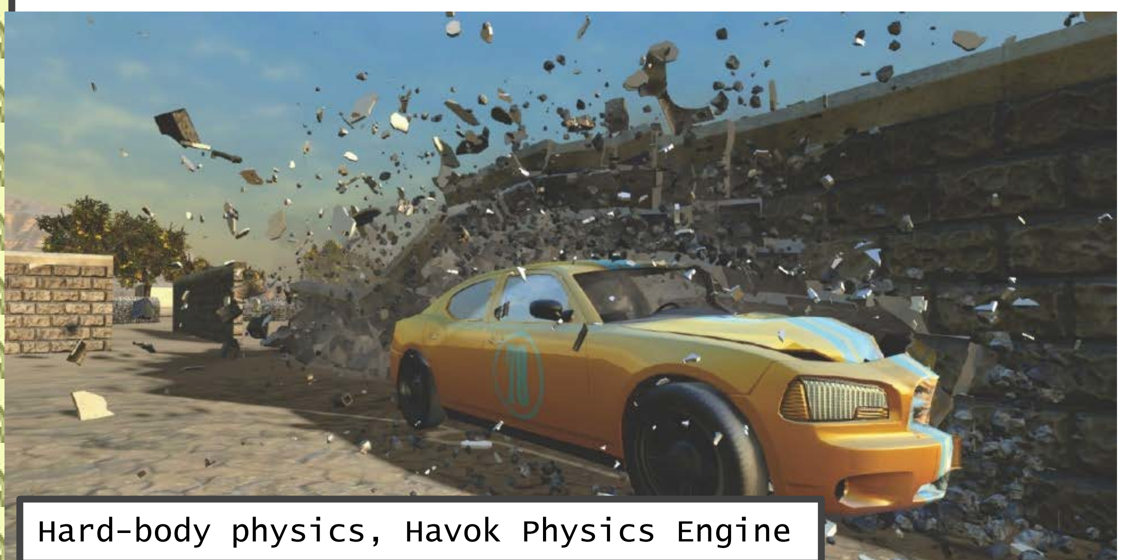
Create some mechanical mayhem as you learn about Unity's physics options.

### 3D Physics

1. Colliders
2. Colliders as Triggers
3. Rigidbodies
4. Adding Physics Forces
5. Adding Physics Torque
6. Physics Materials
7. Physics Joints
8. Detecting Collisions with OnCollisionEnter
9. Raycasting

# Physics engines

**Calculate on-the-fly physics simulations, optimized for a game environment.**



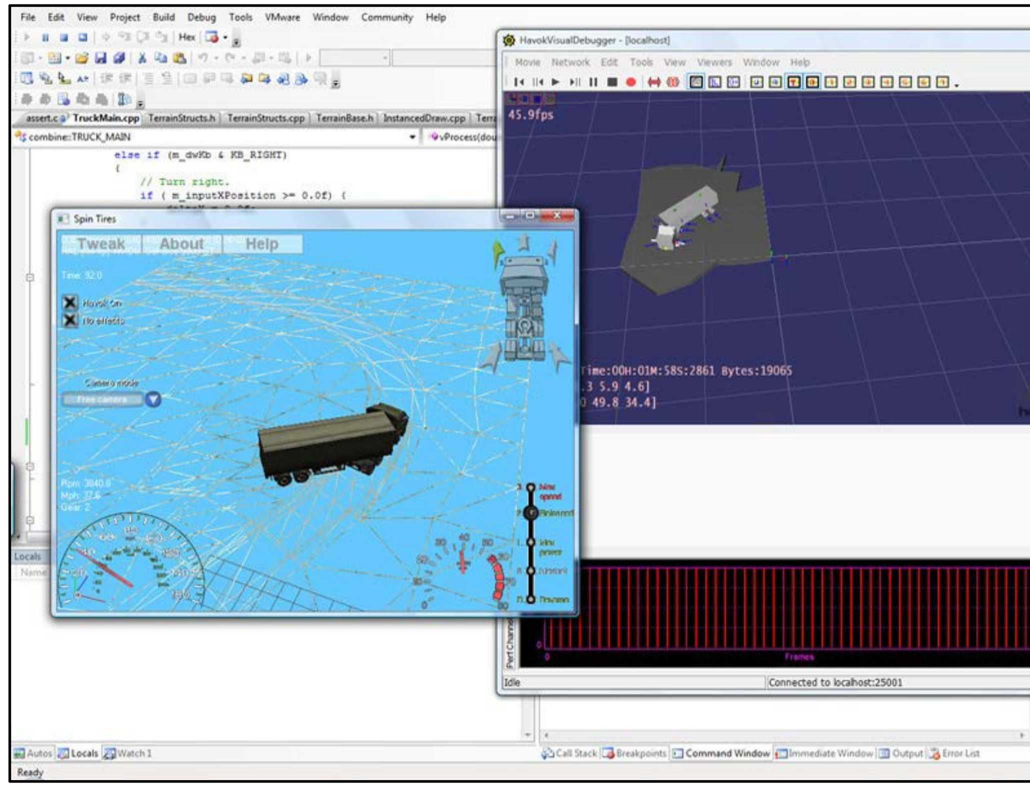
Hard-body physics, Havok Physics Engine


Soft-body physics, CryEngine Physics Engine

# Physics engines



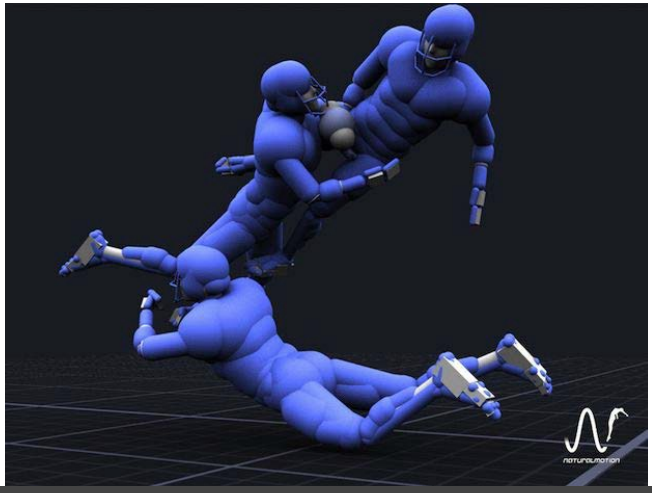
**SDKs with visual debuggers that allow you to run physics simulations on your object to test your code**



# Dynamic animation



**Euphoria by Natural Motion encodes lots of information about human muscles, bones, and nerves to dynamically create realistic character movement like falls.**



# Dynamic animation

**Natural Motion editor with visual programming.**

Art assets & animation

Graphics engines

Physics engines

**Game loop**

# Game loop

update player health

update monster health

physics engine

render scene

sound effects

Heads-up-display

unity

https://unity.com/

A leading game engine

# Guest lecture in 05-830, Spring 2020

- Lead developers at Unity
- Video of the presentation

# A tale of three UI frameworks

A decade of evolving developer and designer workflows in a game engine

**Adam Mechtley**

Lead Developer

DOTS Physics & Rigging

**Damian Campeanu**

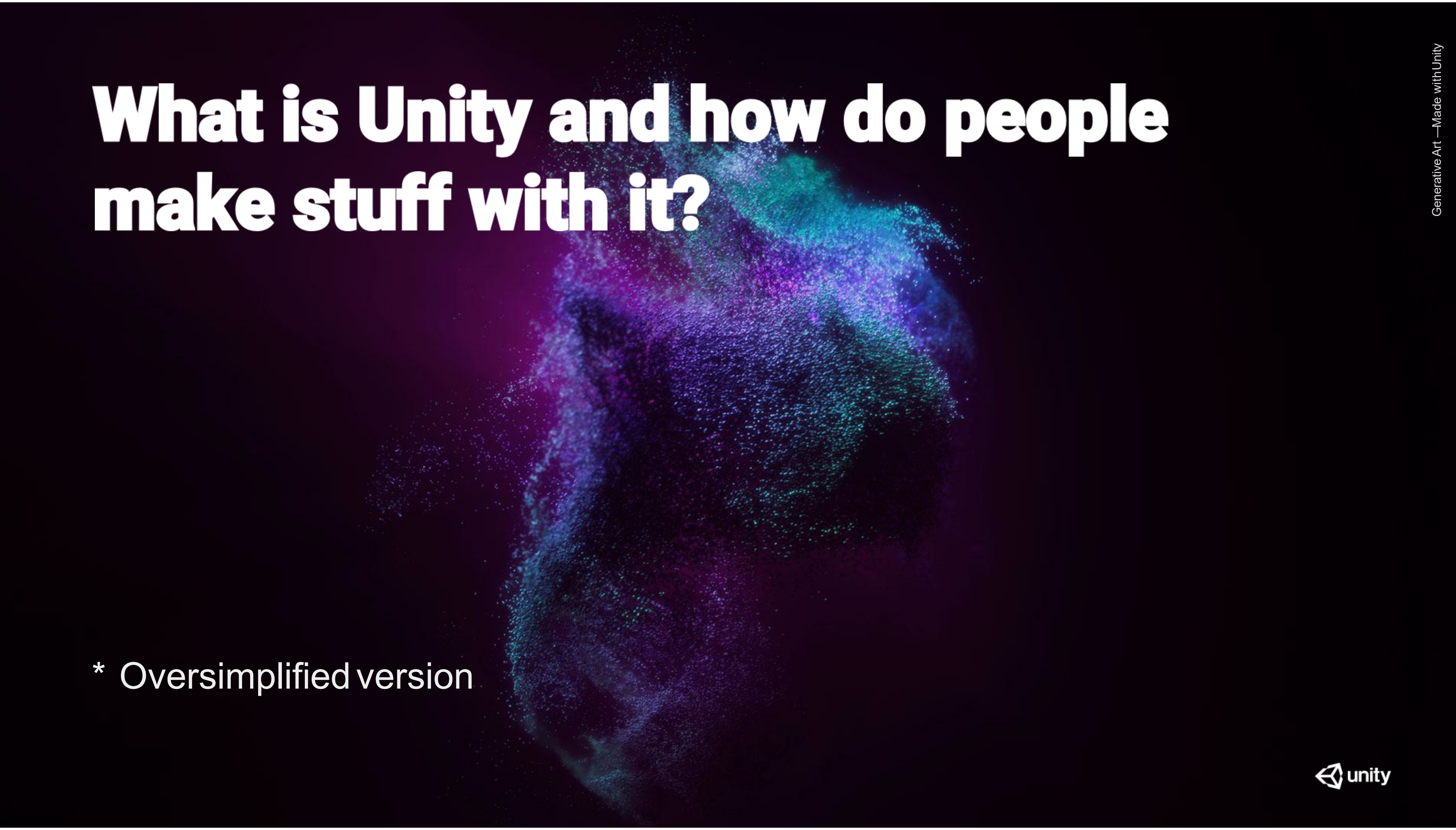
Developer

Editor & UI

unity

Generative Art —Made with Unity

# What is Unity and how do people make stuff with it?




* Oversimplified version

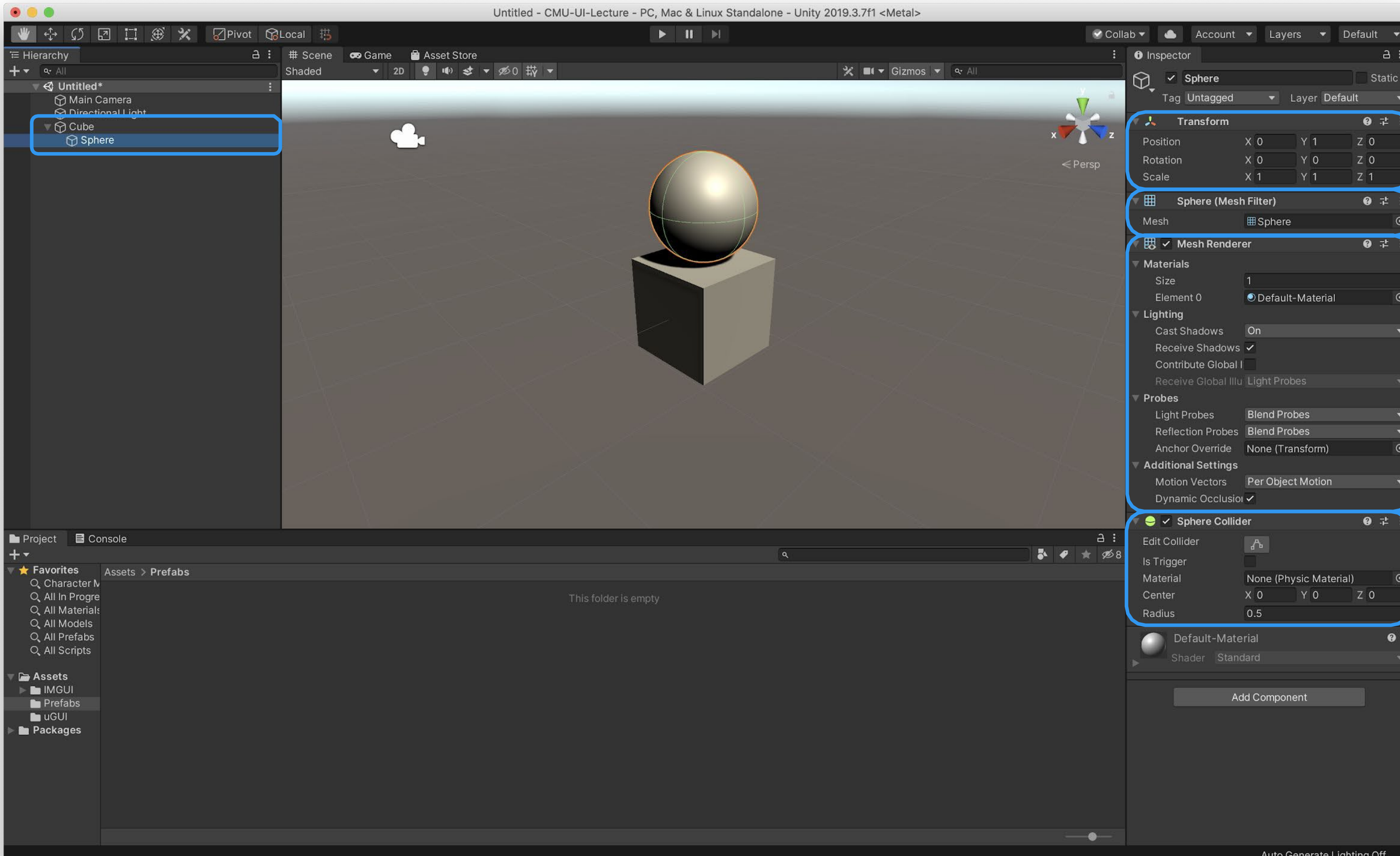
# More than just a game engine!

- Run-Time
  - Built-in libraries (input, animation, physics, UI, rendering, etc.)
  - Compatible with much of .NET ecosystem
- Editor
- Services
  - Analytics, live ops, etc.
- Asset Store
- Millions of users, hundreds of thousands active monthly

# How do users make stuff with Unity?

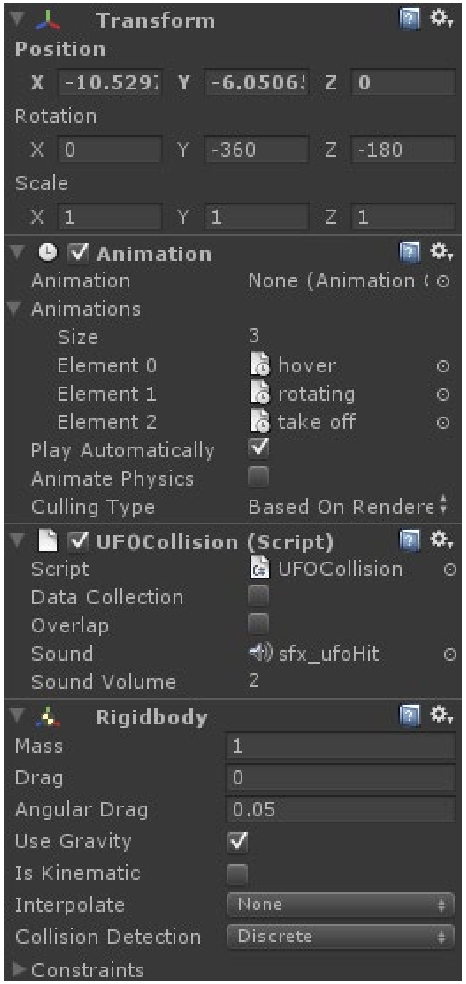


- Create **GameObjects** and add **Components** to produce behavior
- Create new Components via C#
  - Or imported from other programs like AutoDesk's Maya
- Create reusable **Prefabs** from GameObjects
- Prefabs can be nested in each other with overrides

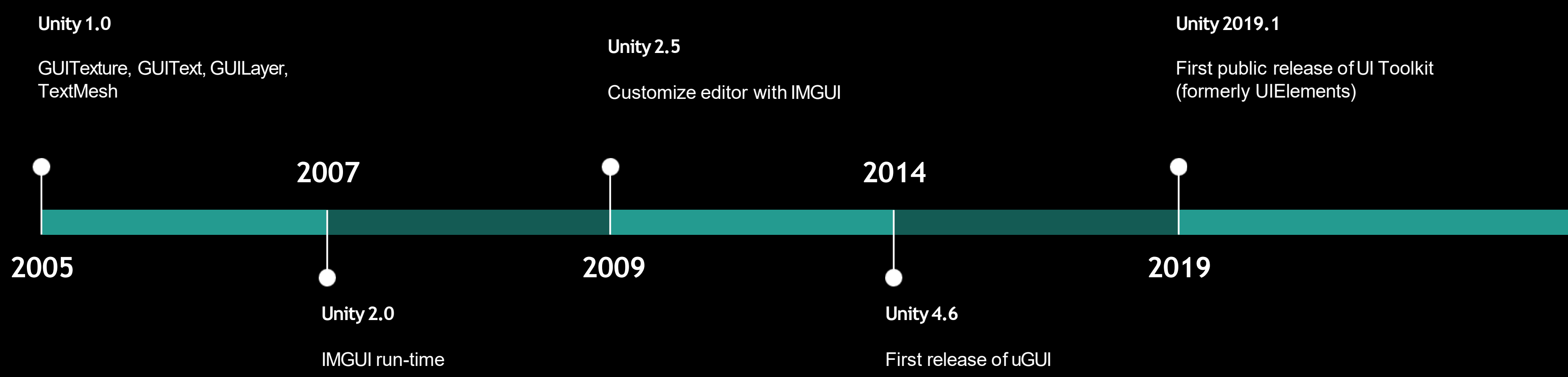

Untitled - CMU-UI-Lecture - PC, Mac & Linux Standalone - Unity 2019.3.7f1 <Metal>
Hierarchy
Untitled*
Main Camera
Directional Light
Cube
Sphere
Scene
Game
Asset Store
Shaded
Gizmos
Persp
Inspector
Sphere
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 1 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sphere (Mesh Filter)
Mesh Sphere
Mesh Renderer
Materials
Size 1
Element 0 Default-Material
Lighting
Cast Shadows On
Receive Shadows
Probes
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Additional Settings
Motion Vectors Per Object Motion
Sphere Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Radius 0.5
Default-Material
Shader Standard
Add Component
Project
Console
Favorites
All Models
All Prefabs
All Scripts
Assets
IMGUI
Prefabs
uGUI
Packages
Assets > Prefabs
This folder is empty
Auto Generate Lighting Off

# Components in Unity

- All public members of a script are exposed in the GUI allowing non-programmer team members to control game settings
- Built in Component types can also be accessed and edited this way
- Properties can be changed in the GUI while the game is running to test changes



# How do users create UI with it?



# IMGUI

A simple UI framework for an ever-changing world

Unity 2.5

Customize editor with IMGUI

2007

2009

Unity 2.0

IMGUI run-time

# Design considerations

- Unity needed a UI framework! (both run-time and editor)
- Most Unity projects…
  - ...were small web player experiences
  - ...were created by small teams with few/broad role specializations
- Most game UI…
  - ...communicate frequently updating values
  - ...overlays

# IMGUI API

- OnGUI() callback
  - Event loop with `Event.current`
  - Call order determines event handling priority
- Library of static methods in GUI class for common functionality
  - GUILayout variants to assist with Rect calculations
  - Both run-time and editor-only variants for most types
- GUIStyle class
- GUISkin asset

# IMGUI advantages and disadvantages

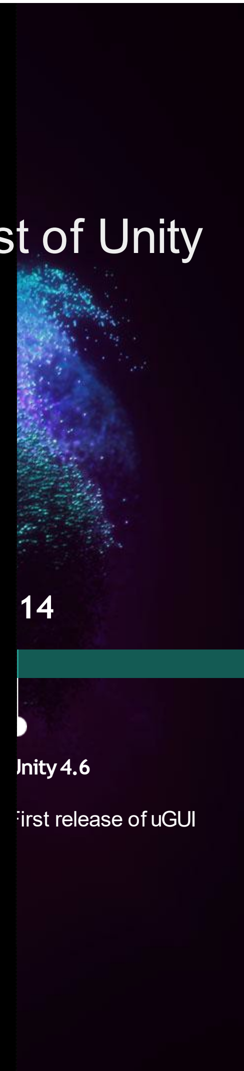
Human-Computer Interaction Institute

+ Gathering and responding to input is trivial
+ Fast for programmers to prototype with
+ Works well for property grids
+ Simple API organization
+ Predictable performance

— But not very great performance
— Limited designer workflows
— No control over rendering pipeline
— Only supports non-diegetic UI
— Lots of manual work making new controls

# uGUI

A framework for to make UI feel more like the rest of Unity



# Design considerations

- Unity needed to empower designers to be productive more independently
- Most Unity projects…
  - ...were created by teams with clearer role specializations
  - ...were run on mobile platforms where draw calls are expensive and display specifications vary wildly
- Most game UI…
  - ...contained in-story/spatial and non-in-story elements
  - ...were richly animated with effects

# uGUI API

- UIBehaviour base class inherits `MonoBehaviour`
- `Selectable`, `Graphic`, etc., sub-classes
- Canvas and CanvasScalar control rendering of hierarchies of elements
  - RectTransform (inherits `Transform`) use for layout
  - Most components draw Sprite assets
  - Set geometry, materials, etc. on child CanvasRenderer
- StandaloneInputModule and EventSystem gather and delegate input events
  - BaseRaycaster of some kind finds event handlers
  - IPointerDownHandler, IPointerUpHandler, IDragHandler, etc.

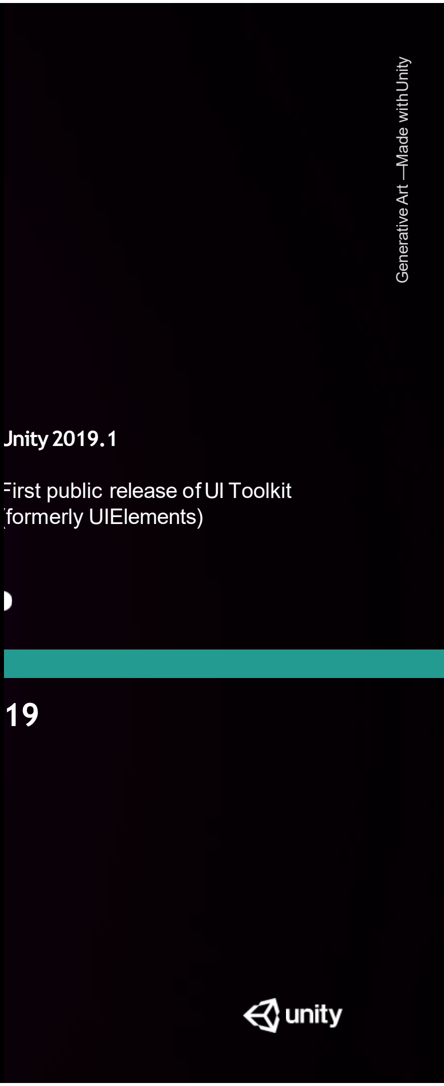
# uGUI advantages and disadvantages

+ Designer workflows that fit with other Unity features (prefabs, animation, etc.)
+ Serializable event handlers
+ Automatic atlasing and scaling based on physical size, DPI, etc.
+ Diegetic UI
+ Common rendering pathway with everything else

— Performance overhead from GameObjects and Components
— Authoring data format hard to read and debug at a glance
— No centralized styling
— Canvases require specialized knowledge to optimize

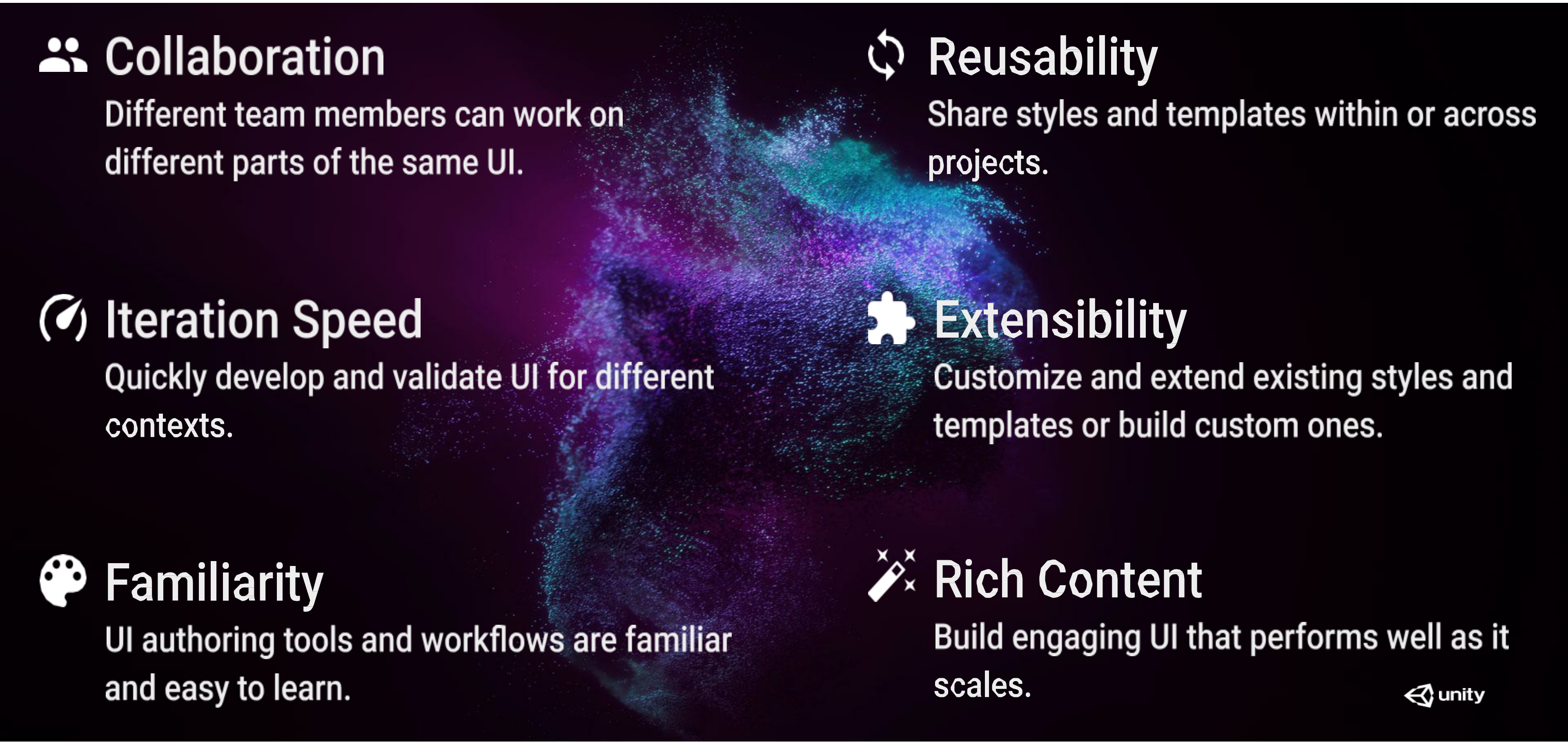
# UI Toolkit

A framework to make Unity feel more like the rest of the world



# Design considerations





# UI Toolkit advantages and disadvantages

- + Great performance for most use cases
- + Powerful automatic layouting via Flexbox
- + Centralized styling using standard paradigms (CSS)
- + Visual authoring without writing code
- + One API for both Editor and Runtime

- — Name-based handles can easily break
- — Inefficient when lots of things are changing at once
- — More complicated Event-based value bindings
- — More complicated bindings to Unity objects and gameplay

# Final thoughts

- Immediate-mode and retained-mode GUI each have strengths and disadvantages in different situations
- As the rest of the world evolves, so, too, must your API
  - Everything comes with a maintenance cost
- Reasonableness of API design decisions is very contextual
  - Aesthetic tastes of the historical moment
  - Technical requirements of target hardware
  - Tools ecosystem
- Design influences users' expressive capabilities