## SI4 - CRÉATION DE MONDES VIRTUELS

# INTERACTIVE 3D APPLICATIONS

Hui-Yin (Helen) Wu

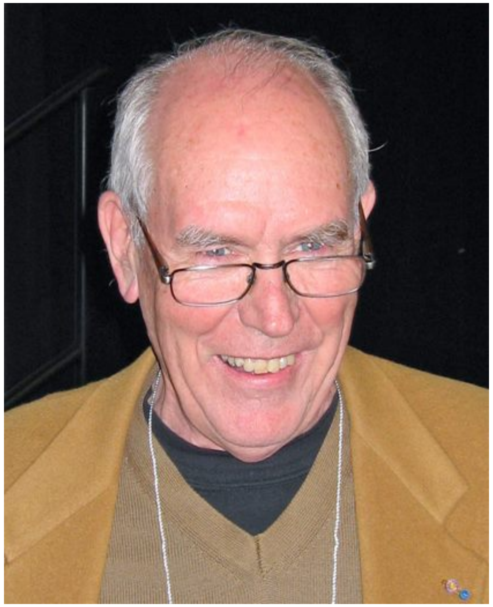
Chargée de recherche (ISFP), Centre Inria d'Université Côte d'Azur

# PLAN

1. Introduction
2. Game engines and architecture
3. Game loop and design patterns
4. Demo of Unity
5. Project and TD4 announcement

# INTRODUCTION

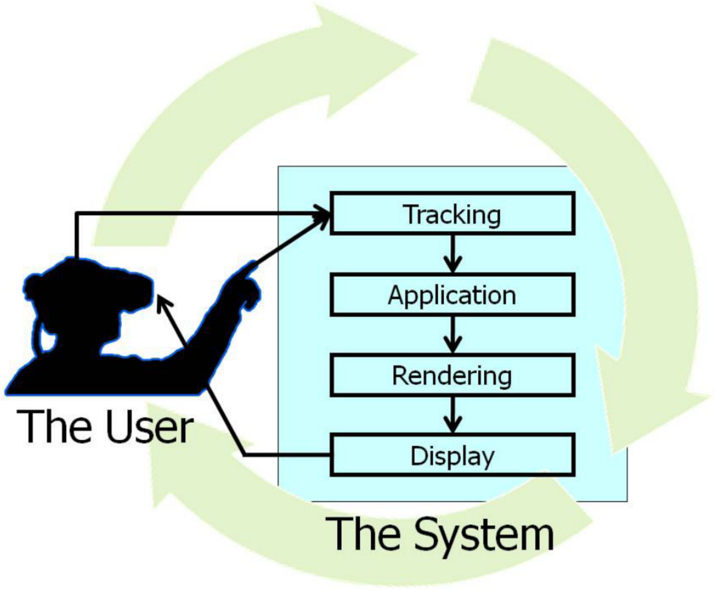
# INTRODUCTION



The ultimate display would, of course, be a room [within which] a chair displayed would be good enough to sit in. *Ivan Sutherland, 1965*

# INTRODUCTION

Reality systems: have for the purpose to effectively communicate the application content to and from the user in an intuitive way as if the user is interacting with the real world.



Jerald, J. (2015). The VR book: Human-centered design for virtual reality. Morgan & Claypool.

# INTRODUCTION

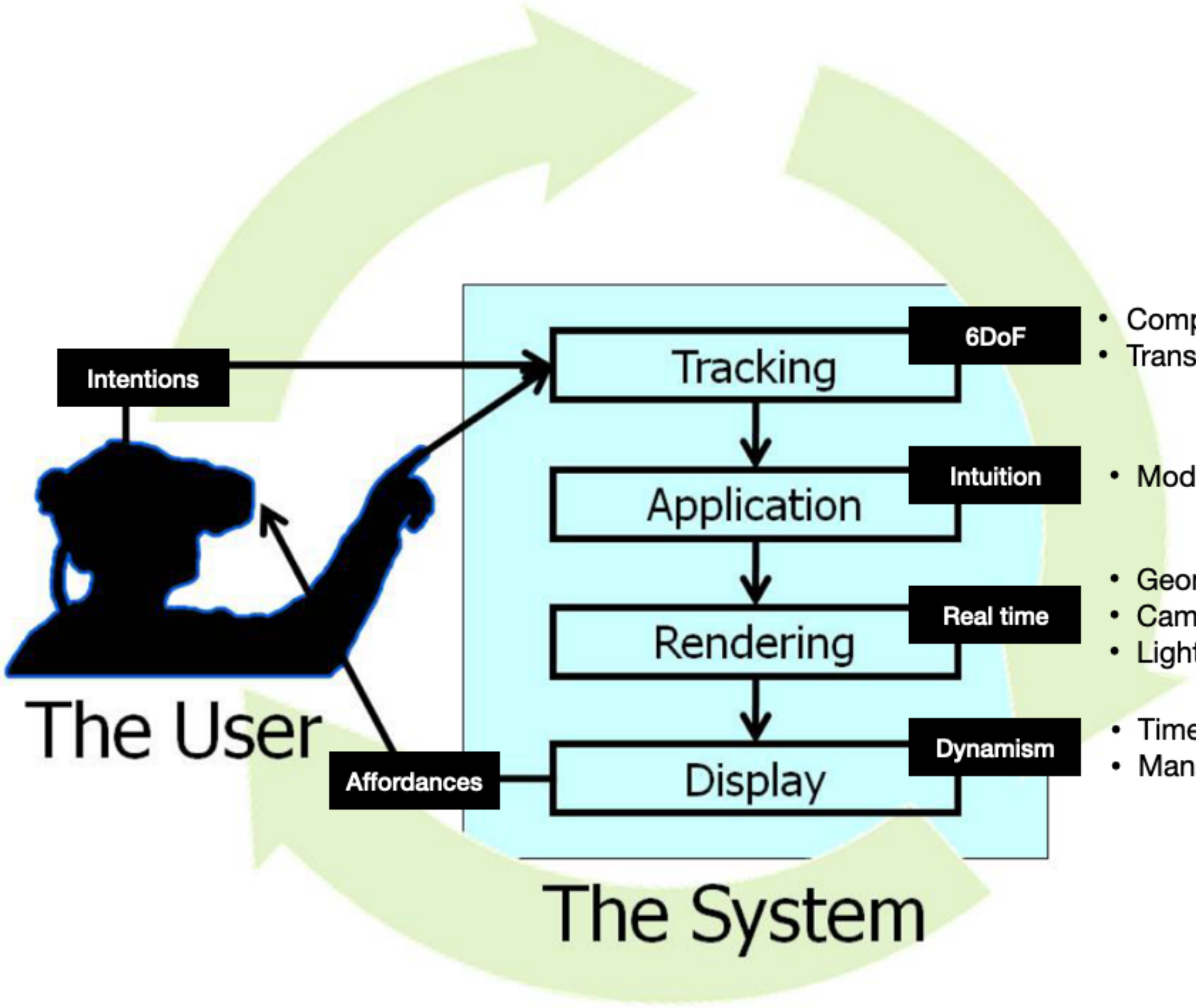
Components of an Immersive Virtual Environment system

- Input: collects data from the user
- Application: includes non-rendering aspects of the virtual world
- Rendering: is the transformation of the internal representation into a user experience
- Display: is the physical representation experienced by the user

Jerald, J. (2015). The VR book: Human-centered design for virtual reality. Morgan & Claypool.

# INTRODUCTION

## Why is it challenging?



- Complexity of potential inputs and intentions
- Transfer functions between 2D and 3D

- Model of the world and its interactive properties

- Geometric forms
- Camera viewpoint
- Lights, physics, and sound

- Timely and adapted response to user input
- Management of latency

# PLAN

1. Introduction
2. Game engines and architecture
3. Game loop and design patterns
4. Demo of Unity
5. Project and TD4 announcement

# GAME ENGINES

A bit of history

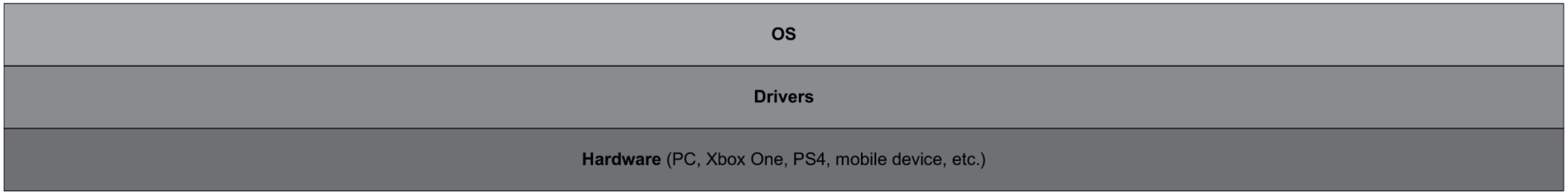
- Atari 2600 (1977): Advent of home gaming consoles and in-house game engines
- c. 1985: Nintendo sidescrolling platforms
- First person shooters (1992): Wolfstein 3D, Quake (1996, namesake), Doom
- Unreal engine (1998): GUI, mods, game logic programming (Blueprints)
- Competition (200X): Valve (Half-life, portal), EA (Frostbite engine), RAGE (GTA, RDR), Crytek (CRYENGINE*), Square Enix (Luminous engine)
- Unity (2005): Game engine as an industry
- Open source engines (2005-): OGRE (2005), GODOT (2014), UPBGE (2019), Open3D (2021)
- XR systems and mobile applications (201X):

* CRYENGINE became Lumberyard after being acquired by Amazon. In 2021, a new open source game engine Open3D was released based on CRYENGINE.

Gregory, J. (2018). Game engine architecture. AK Peters/CRC Press.

# GAME ENGINES

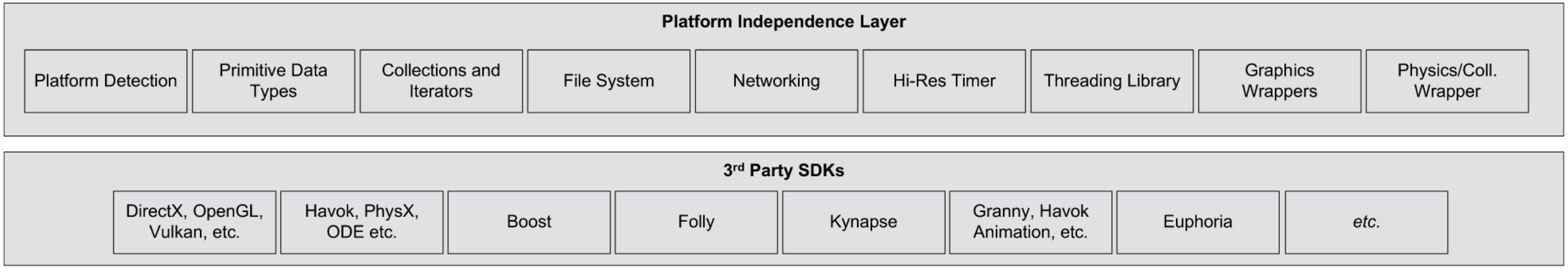
A game engine tries to facilitate the recurring issues one encounters globally, from one application to another.



Gregory, J. (2018). Game engine architecture. AK Peters/CRC Press.

# GAME ENGINES

A game engine tries to facilitate the recurring issues one encounters globally, from one application to another.



**Platform Independence Layer**

Platform Detection · Primitive Data Types · Collections and Iterators · File System · Networking · Hi-Res Timer · Threading Library · Graphics Wrappers · Physics/Coll. Wrapper

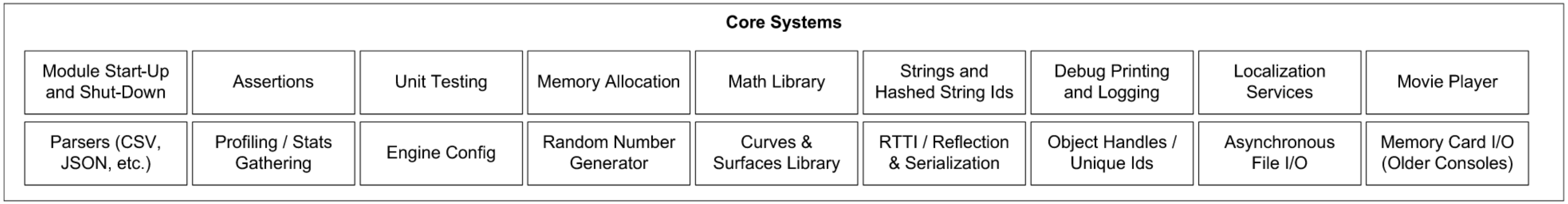
**3rd Party SDKs**

DirectX, OpenGL, Vulkan, etc. · Havok, PhysX, ODE etc. · Boost · Folly · Kynapse · Granny, Havok Animation, etc. · Euphoria · *etc.*

Gregory, J. (2018). Game engine architecture. AK Peters/CRC Press.

# GAME ENGINES

A game engine tries to facilitate the recurring issues one encounters globally, from one application to another.



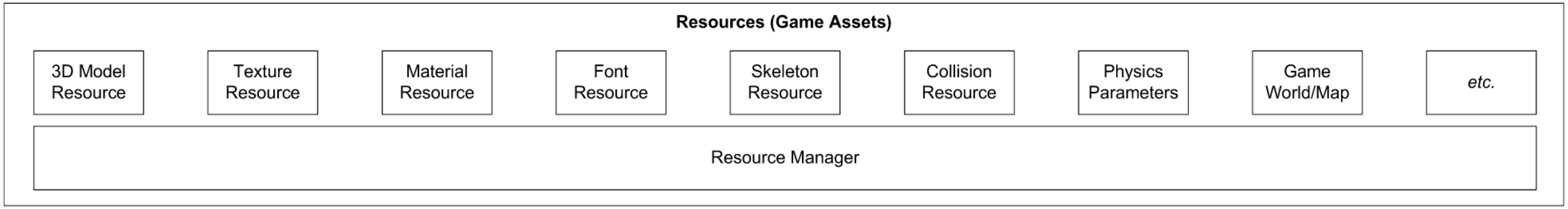
Core Systems

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Module Start-Up and Shut-Down | Assertions | Unit Testing | Memory Allocation | Math Library | Strings and Hashed String Ids | Debug Printing and Logging | Localization Services | Movie Player |
| Parsers (CSV, JSON, etc.) | Profiling / Stats Gathering | Engine Config | Random Number Generator | Curves & Surfaces Library | RTTI / Reflection & Serialization | Object Handles / Unique Ids | Asynchronous File I/O | Memory Card I/O (Older Consoles) |

Gregory, J. (2018). Game engine architecture. AK Peters/CRC Press.

# GAME ENGINES

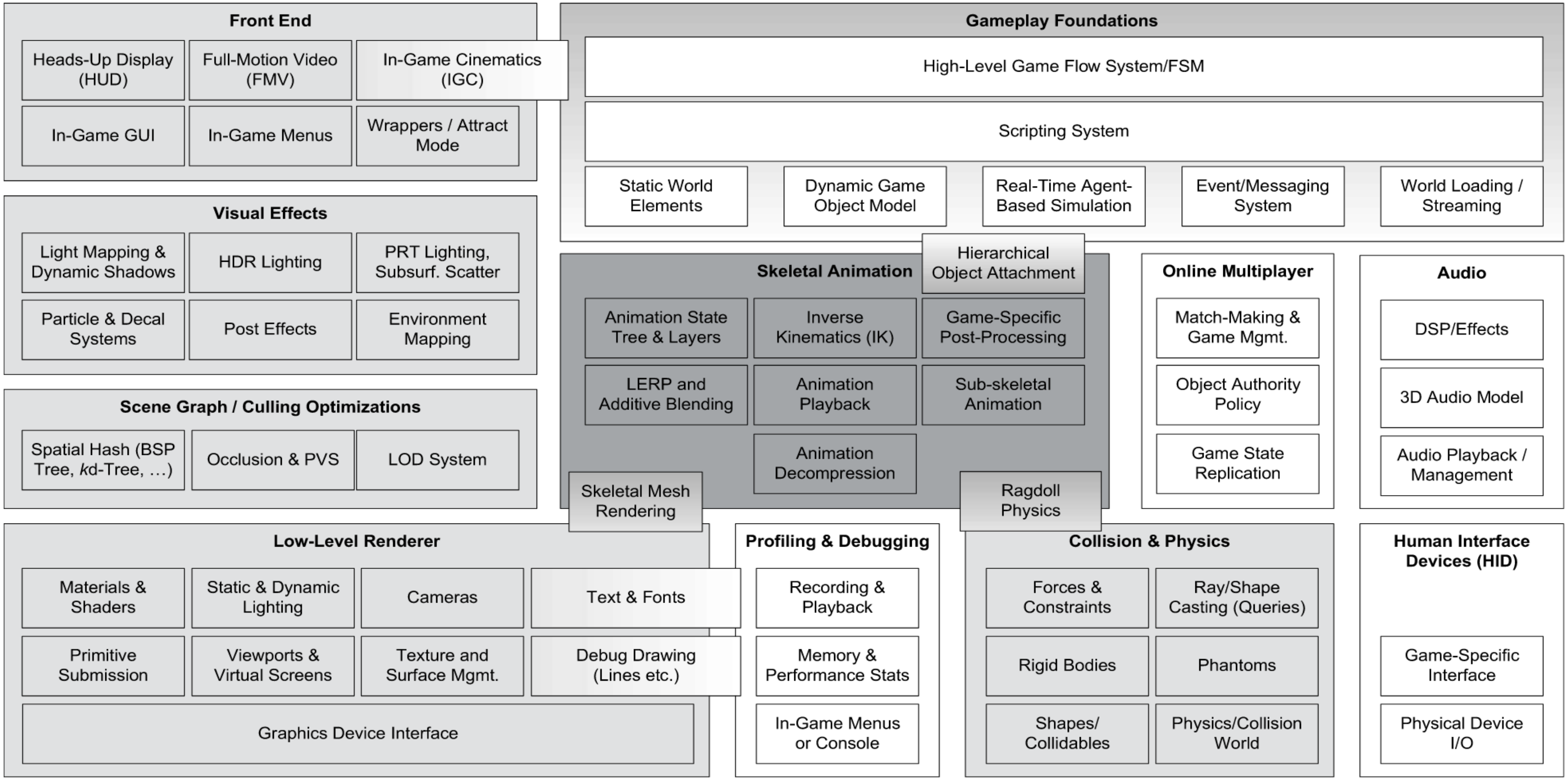
A game engine tries to facilitate the recurring issues one encounters globally, from one application to another.



Gregory, J. (2018). Game engine architecture. AK Peters/CRC Press.

# GAME ENGINES

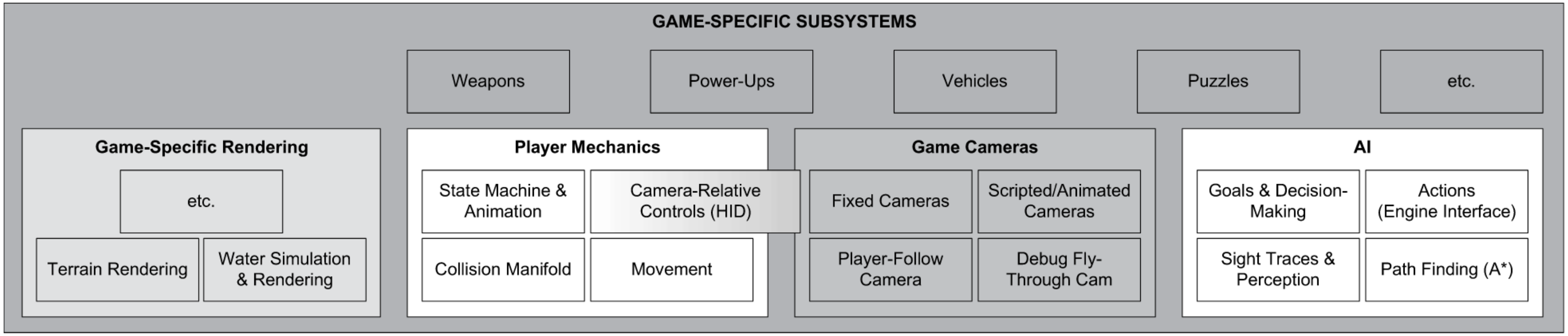
A game engine tries to facilitate the recurring issues one encounters globally, from one application to another.



Gregory, J. (2018). Game engine architecture. AK Peters/CRC Press.

# GAME ENGINES

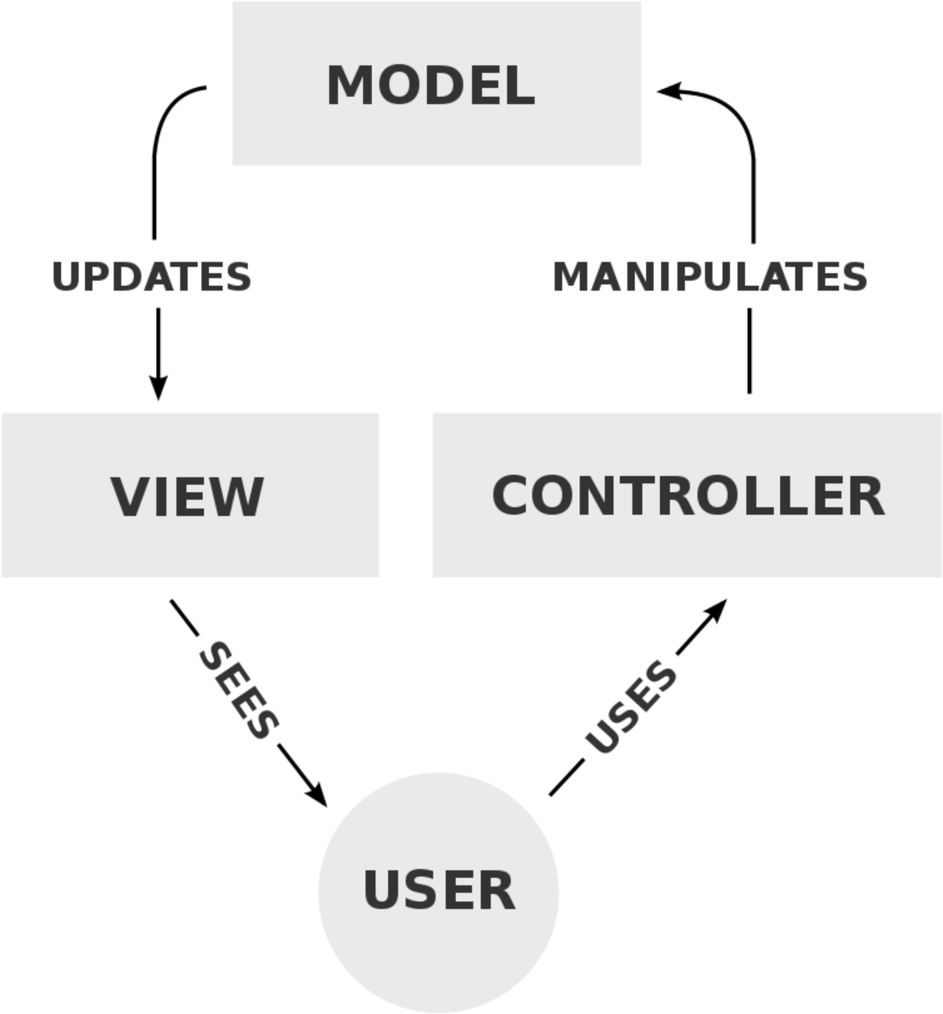
A game engine tries to facilitate the recurring issues one encounters globally, from one application to another.



Gregory, J. (2018). Game engine architecture. AK Peters/CRC Press.

# GAME ENGINES

Software architecture: Model-View-Control (MVC)



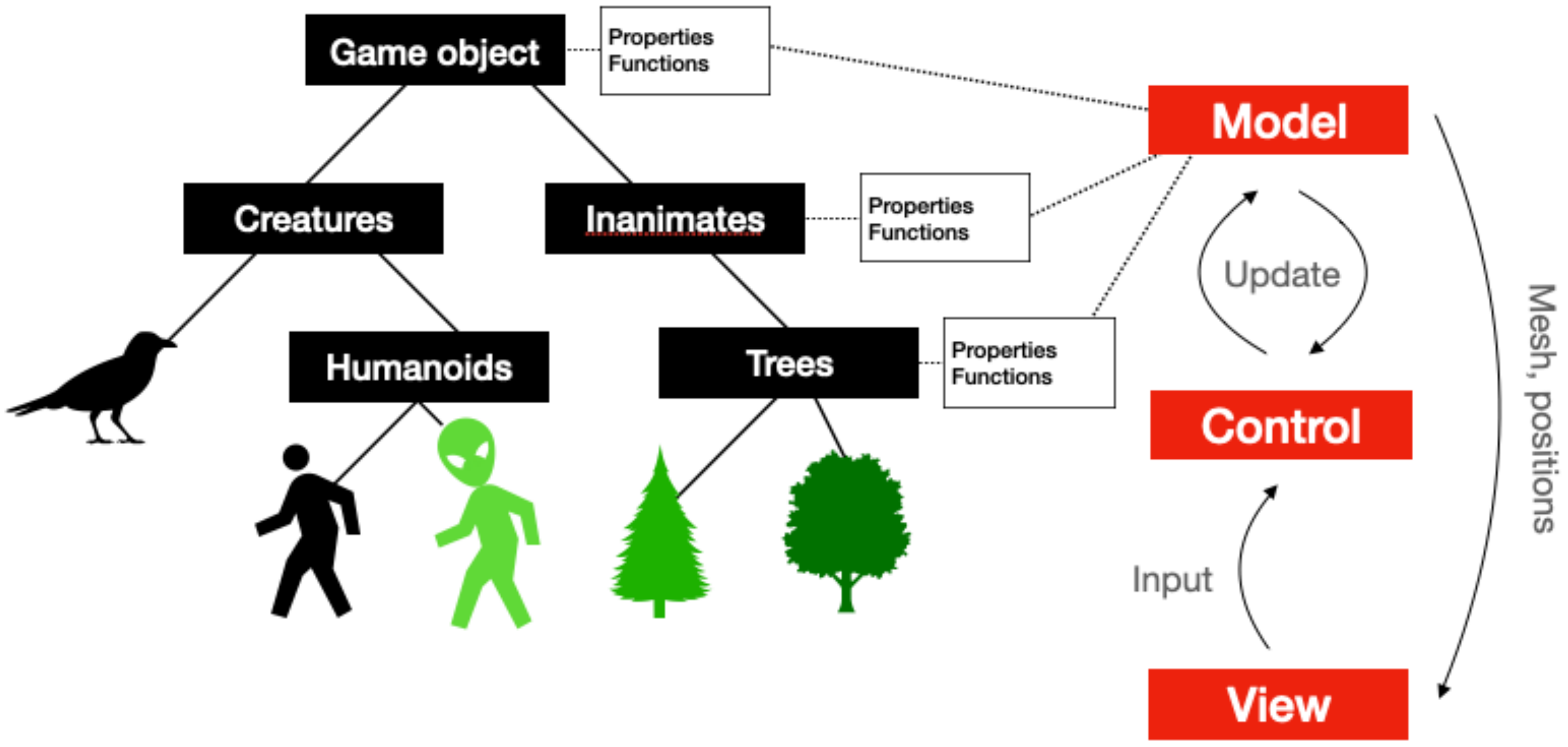
# GAME ENGINES

Software architecture: Model-View-Control (MVC)

- Model: Database, workflows and application state
- View: Input response and rendering
- Control: Input event management and synchronizing model and view

# GAME ENGINES

Model-View-Control (MVC) with OOP approach



- complex inheritance heirarchies (costly lookups)
- memory locality (dynamic allocation, cache misses, loading too much data into RAM)
- inefficient (iterating over all data of all entities individually)

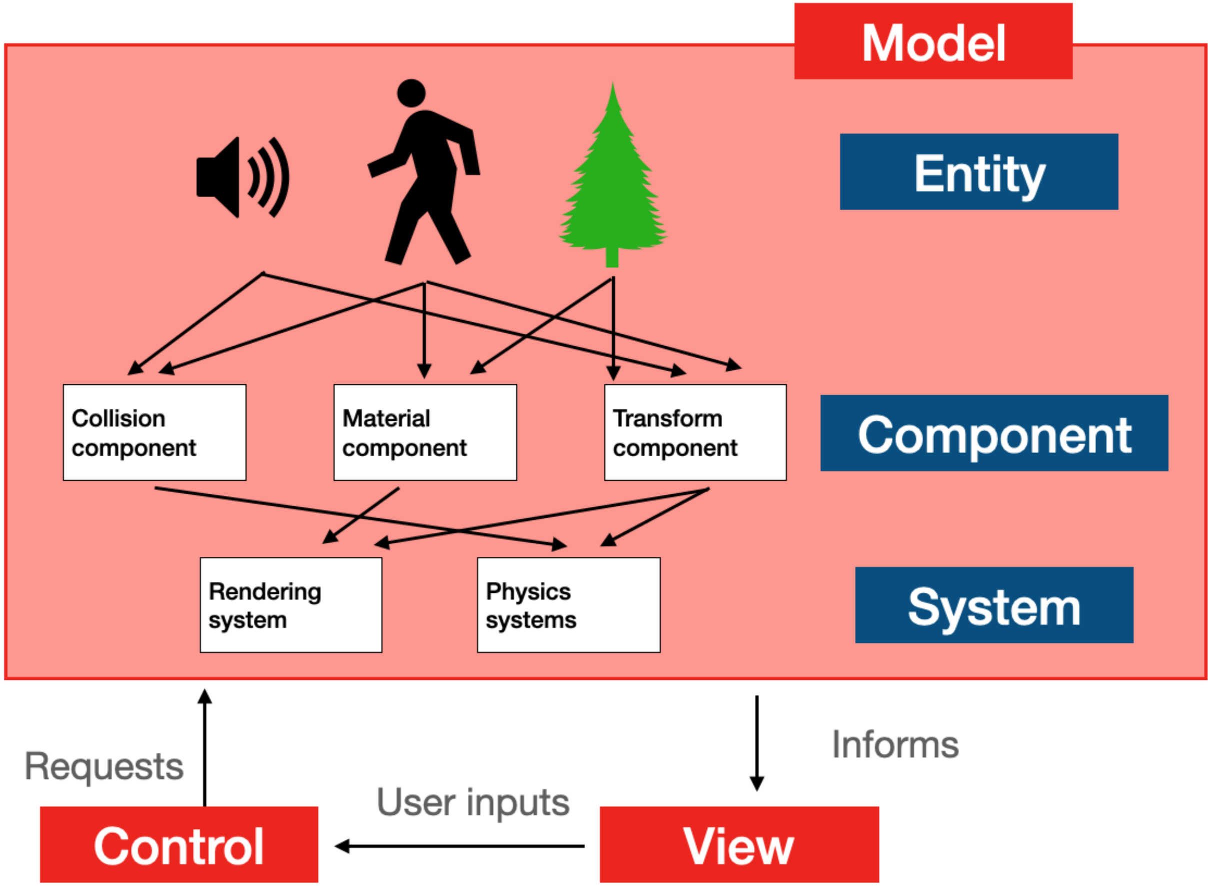
# GAME ENGINES

Entity Component System (ECS): addressing limitations of OOP approach

- Entity: Game objects such as 3D models, camera, light
- Component: object properties, such as transforms, colliders, materials
- System: physics, rendering, AI

# GAME ENGINES

Model-View-Control (MVC) with ECS approach



# PLAN

1. Introduction
2. Game engines and architecture
3. Game loop and design patterns
4. Demo of Unity
5. Project and TD4 announcement

# GAME LOOP AND DESIGN PATTERNS

General question: How long did it take you to render 1 frame in Blender?

# GAME LOOP AND DESIGN PATTERNS

Game update loop

- initialization (once)
- physics (~20Hz)
- interactions
- events, game state updates, AI (4-20 Hz)
- rendering (every frame)
- closing (once)

# GAME LOOP AND DESIGN PATTERNS

In short: Time is of essence in interactive applications. Efficient ways to store and access data, and to communicate are therefore key to good application design.

# GAME LOOP AND DESIGN PATTERNS

Examples

- Responsiveness (for every interaction, there is a timely response)
- Message passing (alerting all entities of interest and only those of interest)
- Resource allocation (parallelisation of code and good memory allocation)

# GAME LOOP AND DESIGN PATTERNS

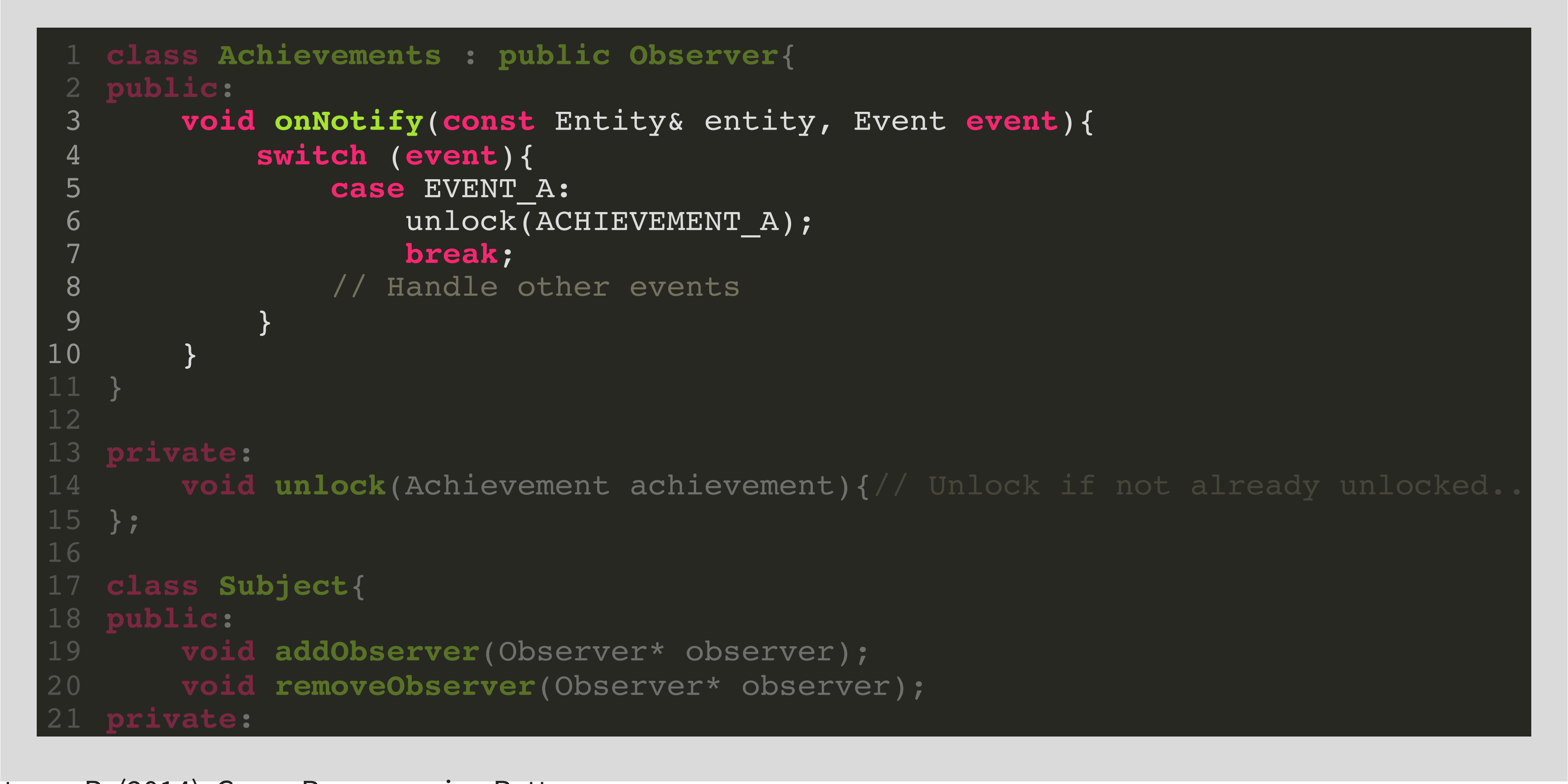
Examples of common game design patterns:

- Responsiveness: Observer
- Message passing: Event queue
- Memory allocation: Flyweight

Nystrom, R. (2014). Game Programming Patterns.

# GAME LOOP AND DESIGN PATTERNS

## Observer

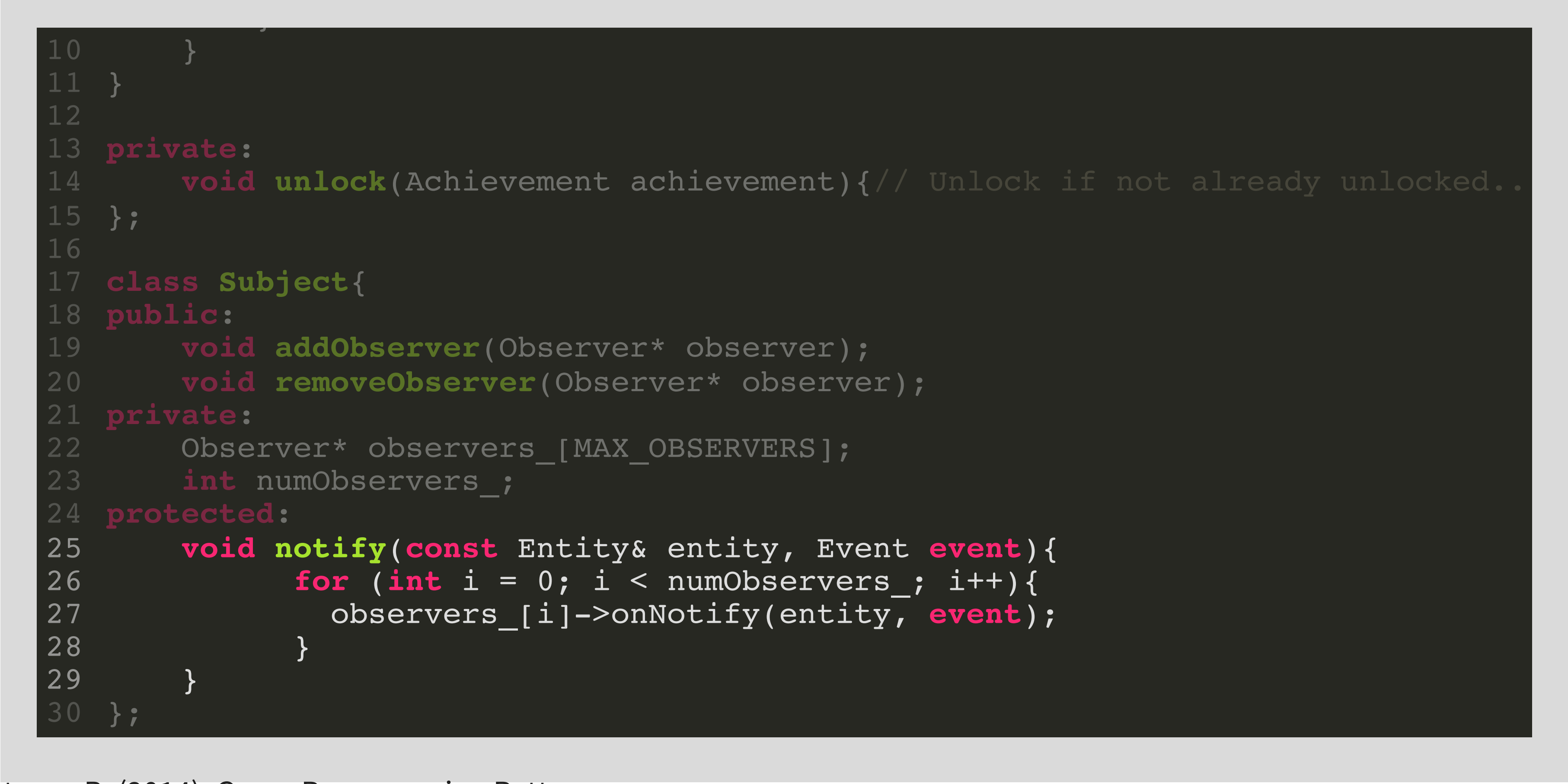


```
class Achievements : public Observer{
public:
    void onNotify(const Entity& entity, Event event){
        switch (event){
            case EVENT_A:
                unlock(ACHIEVEMENT_A);
                break;
            // Handle other events
        }
    }
}

private:
    void unlock(Achievement achievement){// Unlock if not already unlocked..
};

class Subject{
public:
    void addObserver(Observer* observer);
    void removeObserver(Observer* observer);
private:
```

Nystrom, R. (2014). Game Programming Patterns.

# GAME LOOP AND DESIGN PATTERNS

## Observer

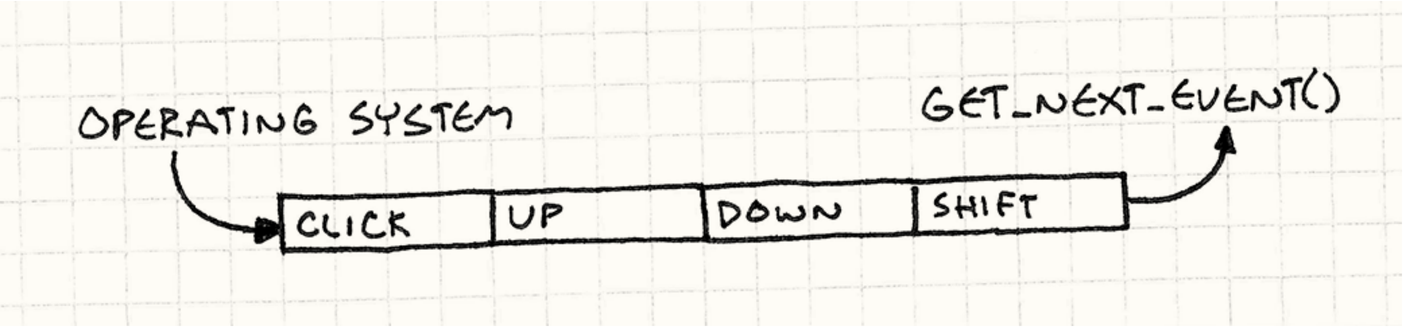


```
    }
}

private:
    void unlock(Achievement achievement){// Unlock if not already unlocked..
};

class Subject{
public:
    void addObserver(Observer* observer);
    void removeObserver(Observer* observer);
private:
    Observer* observers_[MAX_OBSERVERS];
    int numObservers_;
protected:
    void notify(const Entity& entity, Event event){
          for (int i = 0; i < numObservers_; i++){
            observers_[i]->onNotify(entity, event);
          }
    }
};
```

Nystrom, R. (2014). Game Programming Patterns.

# GAME LOOP AND DESIGN PATTERNS

Event Queue



Nystrom, R. (2014). Game Programming Patterns.

# GAME LOOP AND DESIGN PATTERNS

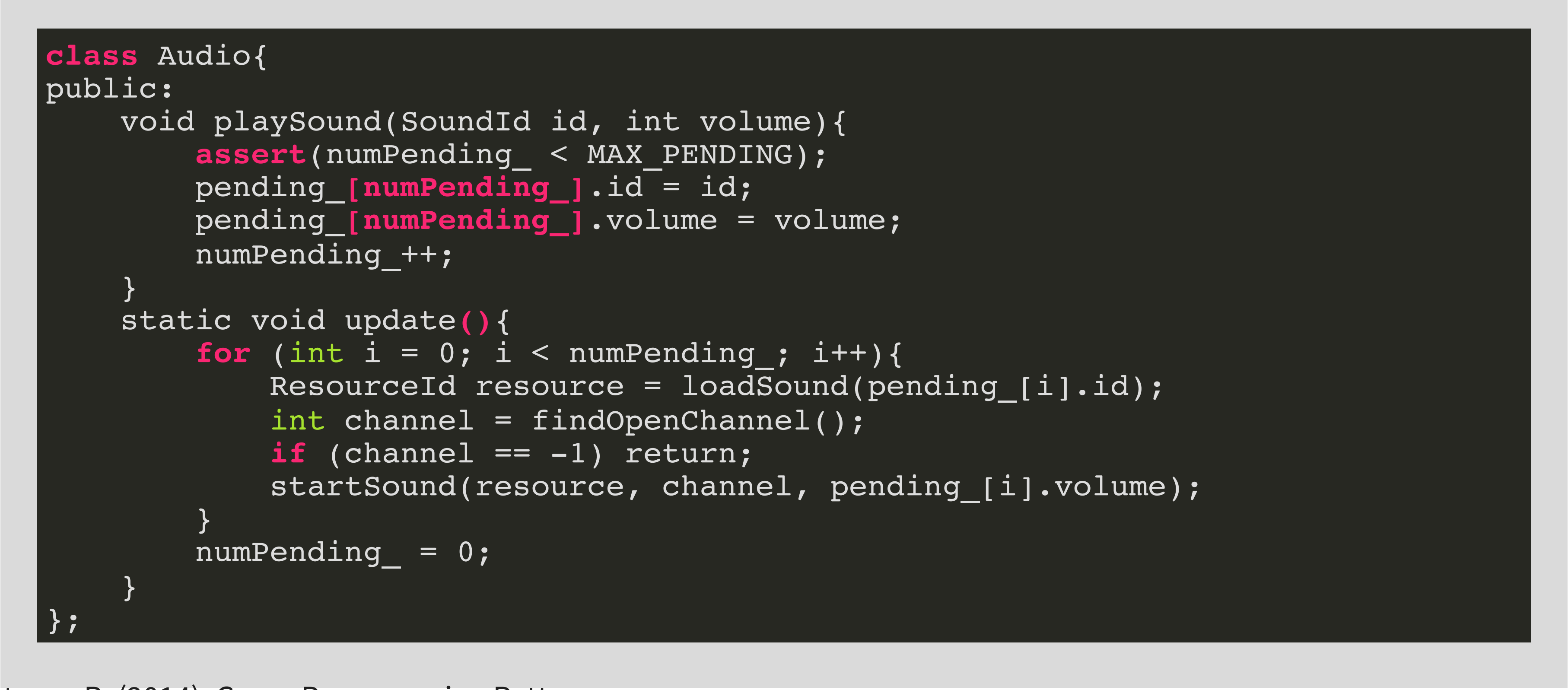
## (Without) Event Queue

```
class Audio{
public:
    static void playSound(SoundId id, int volume);
};

class Menu{
public:
    void onSelect(int index){
        Audio::playSound(SOUND_BLOOP, VOL_MAX);
    }
};
```

Nystrom, R. (2014). Game Programming Patterns.

# GAME LOOP AND DESIGN PATTERNS

Event Queue (also facilitates parallelization and multi-threading)

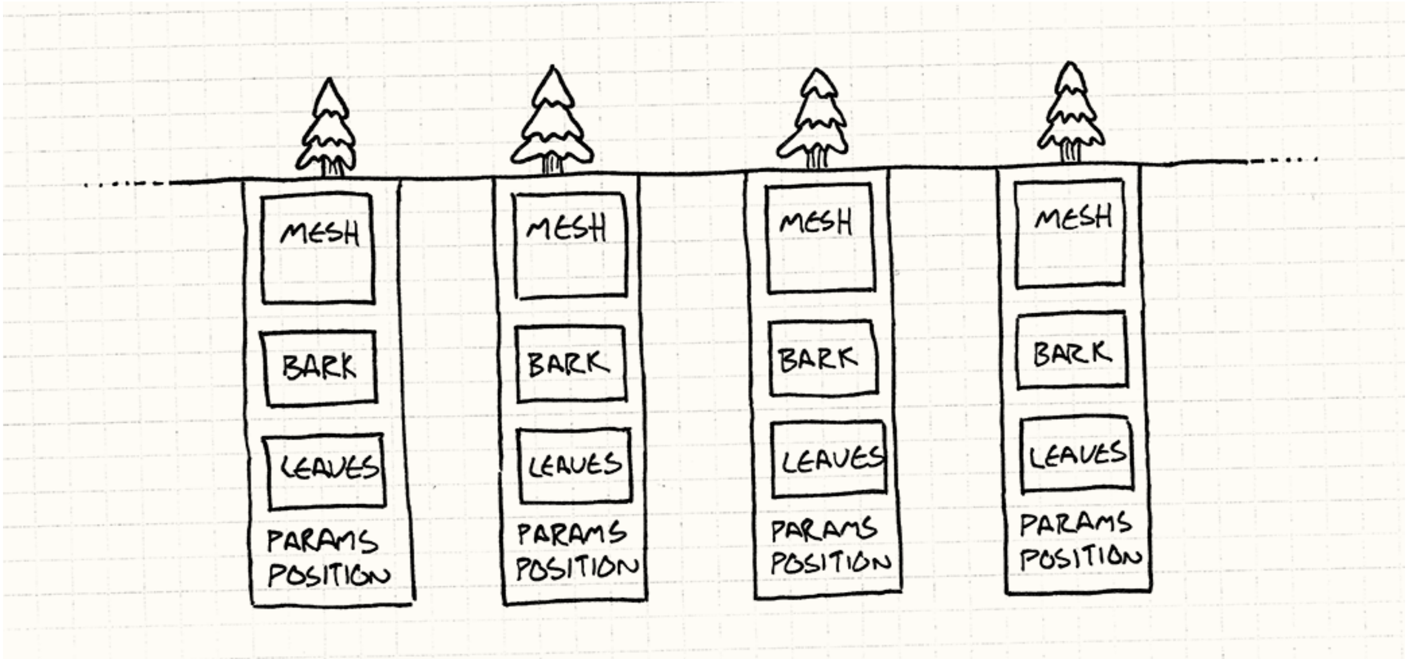


```
class Audio{
public:
    void playSound(SoundId id, int volume){
        assert(numPending_ < MAX_PENDING);
        pending_[numPending_].id = id;
        pending_[numPending_].volume = volume;
        numPending_++;
    }
    static void update(){
        for (int i = 0; i < numPending_; i++){
            ResourceId resource = loadSound(pending_[i].id);
            int channel = findOpenChannel();
            if (channel == -1) return;
            startSound(resource, channel, pending_[i].volume);
        }
        numPending_ = 0;
    }
};
```

Nystrom, R. (2014). Game Programming Patterns.

# GAME LOOP AND DESIGN PATTERNS

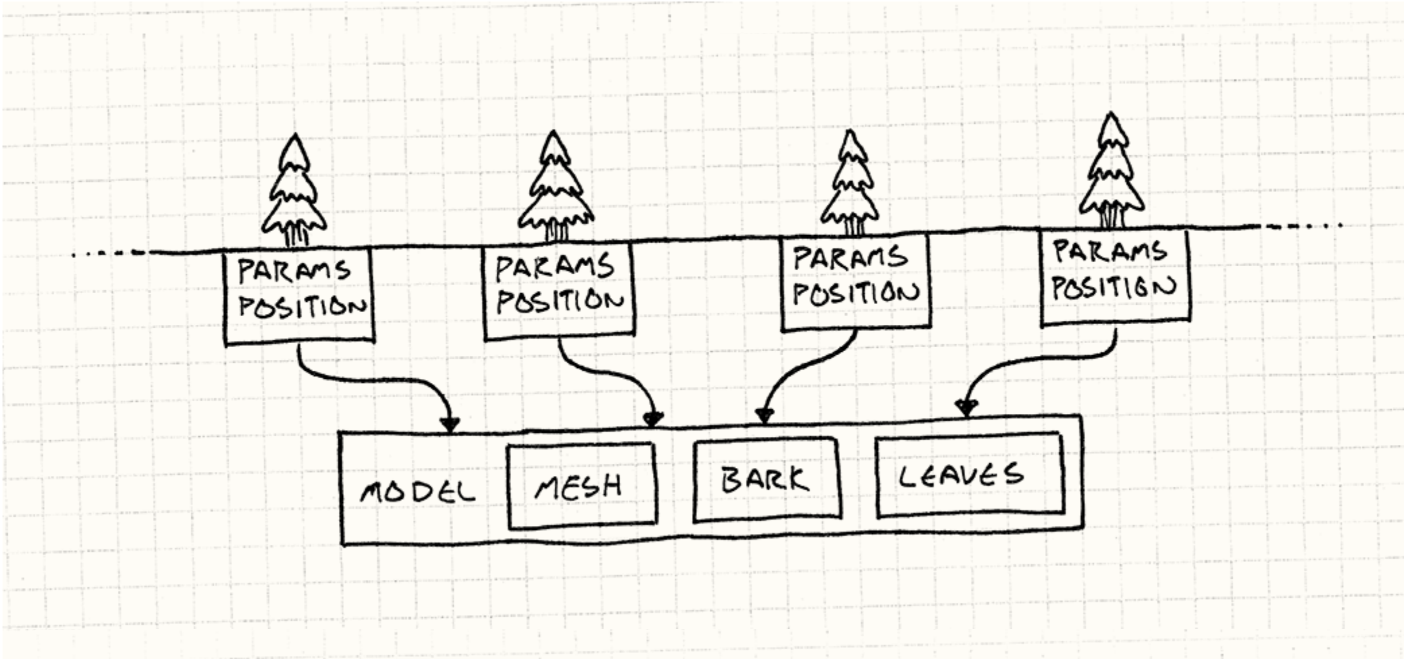
Flyweight



Nystrom, R. (2014). Game Programming Patterns.

# GAME LOOP AND DESIGN PATTERNS

Flyweight



Nystrom, R. (2014). Game Programming Patterns.

# GAME LOOP AND DESIGN PATTERNS

## First approach

```
class Tree{
private:
    Mesh mesh_;
    Texture bark_;
    Texture leaves_;
    Vector position_;
    double height_;
    double thickness_;
    Color barkTint_;
    Color leafTint_;
};
```

Nystrom, R. (2014). Game Programming Patterns.

# GAME LOOP AND DESIGN PATTERNS

Flyweight: a better approach

```
class TreeModel{
private:
    Mesh mesh_;
    Texture bark_;
    Texture leaves_;
};

class Tree{
    TreeModel* model_;

    Vector position_;
    double height_;
    double thickness_;
    Color barkTint_;
    Color leafTint_;
};
```

Nystrom, R. (2014). Game Programming Patterns.

# GAME LOOP AND DESIGN PATTERNS

## Extended reading

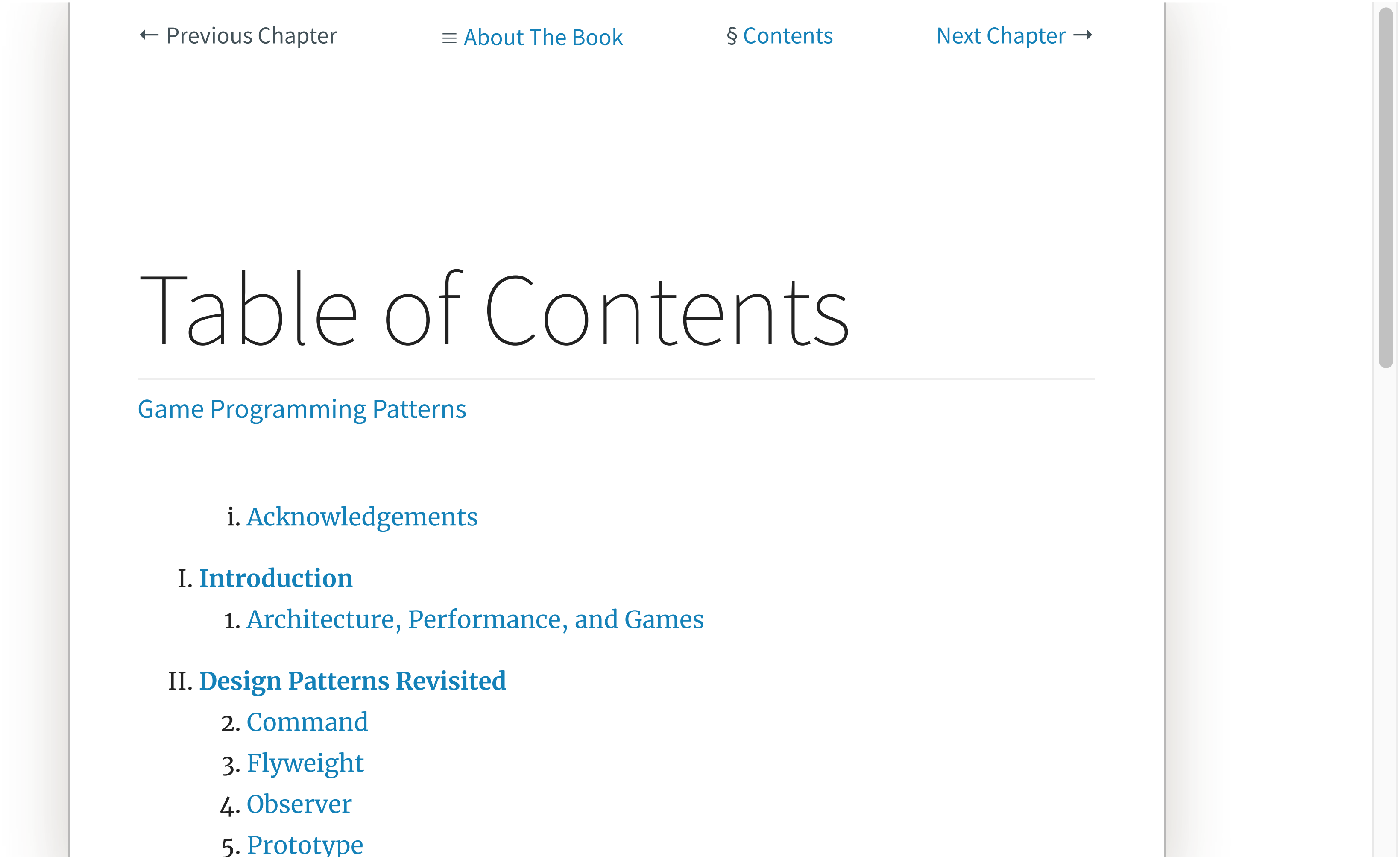


Table of Contents

Game Programming Patterns

Nystrom, R. (2014). Game Programming Patterns.

# PLAN

1. Introduction
2. Game engines and architecture
3. Game loop and design patterns
4. Demo of Unity
5. Project and TD4 announcement

# DEMO OF UNITY

Implementation of an ECS architecture

- Unity user interface (viewport, gizmo, assets, scene heirarchy)
- Primitives
- Components
  - transforms (translation, rotation)
  - materials
- Scripting
- Events

# PLAN

1. Introduction
2. Game engines and architecture
3. Game loop and design patterns
4. Demo of Unity
5. Project and TD4 announcement

# PROJECT

La Petite Princesse

The Little Princess is coming to Polytech from Asteroid EIEIIH8. She is curious and excited to visit and learn more about humanity. On her way, she will visit a number of planets.

Your mission is to design an interactive application to show the little princess a bit more about us during her travels through a series of planets.

# PROJECT

La Petite Princesse - Planets

- Planet 1: The museum of form and motion (TD1-3)
- Planet 2: The museum of interaction and competition (TD4-6)
- Planet 3-X: Of your creation! ( if N = [number of people in your team], then N+2 >= X >= N of people in your team)

# PROJECT

La Petite Princesse - Rules

- Groups of 3-5 within the same TD group
- You need to use Unity for your project
- You may use external assets (models, animations, sounds... etc.), but cite them!
- No sex, no violence, no discrimination, direct or otherwise implied

# PROJECT

La Petite Princesse - Timeline

- TD4-6: implementation of Planets 1-2, plan planets 3+
- TD7: Game design document presentation (5 mins per group)
- TD8-10: Planets 3+
- TD11: Presentation