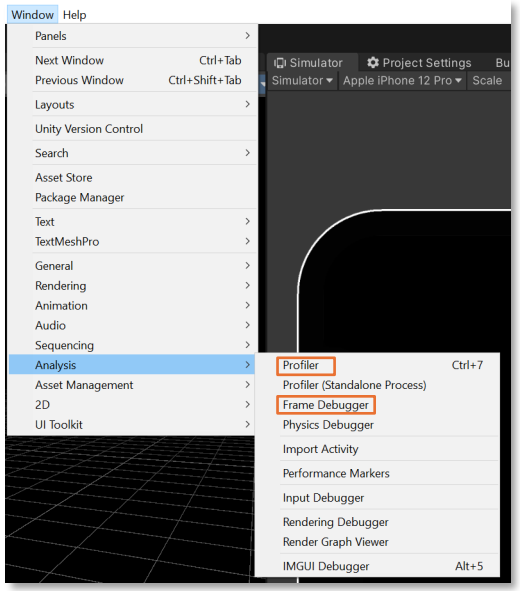
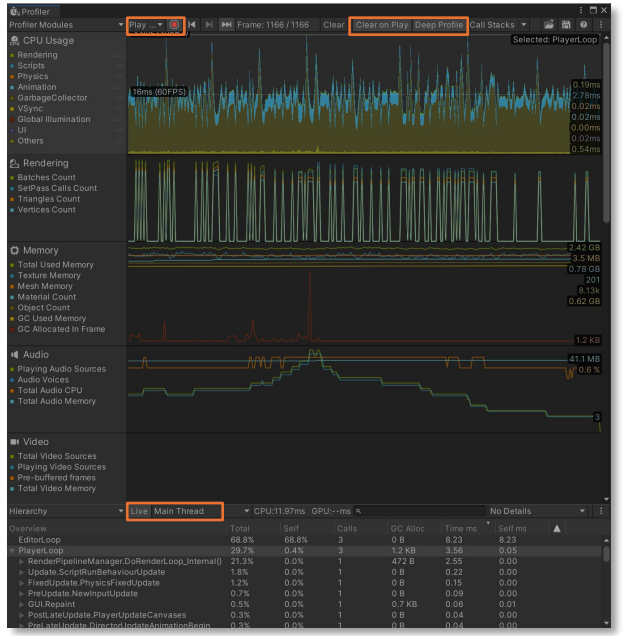
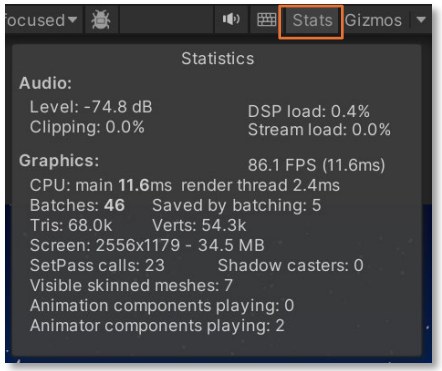
Unity optimizations

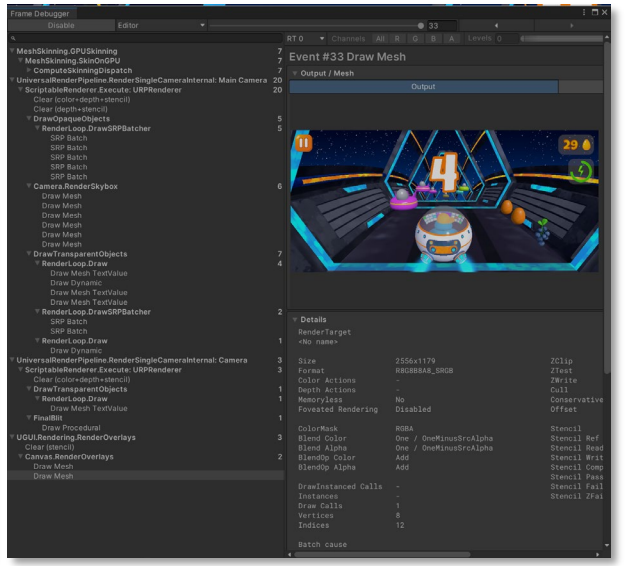
Engine loop

- Per frame
  - Update: computations
  - Render: draw graphics
  - 60fps = 16.7ms to compute and draw
    - Application.targetFrameRate = 60
- Profiler
  - Enable “live” and “deep profile”
  - Examine “PlayerLoop” to determine bottlenecks
- Frame debugger
  - Scrub through event sequence for drawing to screen









Slow down

- Update loop runs at max achievable fps
- Not all checks need to be done at full fps
- Reduce some checks' frequency with timers

```
public class SomeClass : MonoBehaviour
{
  float checkTime = 3f;
  float curCheckTime = 0f;

  private void Update() {
    curCheckTime += Time.deltaTime;
    if (curCheckTime >= checkTime) {
      curCheckTime = 0f;
      PerformSomeCheck();
    }
  }

  private void PerformSomeCheck() {
    // some check
  }
}
```

```
public class SomeClass : MonoBehaviour
{
  int interval = 3;

  private void Update() {
    if (Time.frameCount % interval == 0) {
      // check, skip, skip, check, skip ...
      PerformSomeCheck();
    }
  }
}
```

Caching

- Slow functions
  - Find
  - FindObjectsOfType()
  - GetComponent<>()
  - GetComponentInChildren<>()
- Cache whenever possible
  - Camera
  - Animator
  - Rigidbody
  - Collider
  - MeshRenderer
  - Material

```
public class SomeClass : MonoBehaviour
{
  private Camera mainCamera;
  private Rigidbody rb;
  private Collider collider;
  private TextMeshPro txtScore;
  private Transform txtScoreParent;

  private void Awake() {
    mainCamera = Camera.main;
    rb = GetComponent<Rigidbody>();
    collider = GetComponent<Collider>();
    txtScoreParent = transform.Find(“NameInHierarchy”);
    if (txtScoreParent != null) {
      txtScore = txtScoreParent.GetComponent<TextMeshPro>();
    }
  }

  private void Move() {
    GetComponent<Rigidbody>().AddForce(...);
    rb.AddForce(...);
  }

  private void LateUpdate() {
    mainCamera.transform.position = ...;
  }

  private void UpdateScore(int score) {
    if (txtScore != null) {
      txtScore.text = score.ToString(“d6”);
    }
  }
}
```

Design patterns

- Reusable solutions to commonly occurring problems
- Singleton
- Observer
- Object pooling

Singleton

- Global static access
- Class is persistent across scenes
- Only one instance of class exists across game
- Game manager, ad manager, audio manager

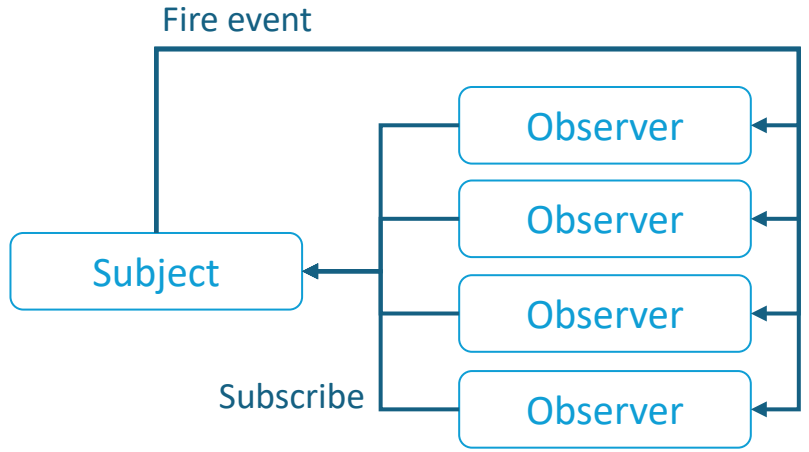
```
public class GameManager : MonoBehaviour
{
  public static GameManager instance;

  private void Awake() {
    if (instance == null) {
      instance = this;
      DontDestroyOnLoad(gameObject);
    } else {
      Destroy(gameObject);
    }
  }
}
```

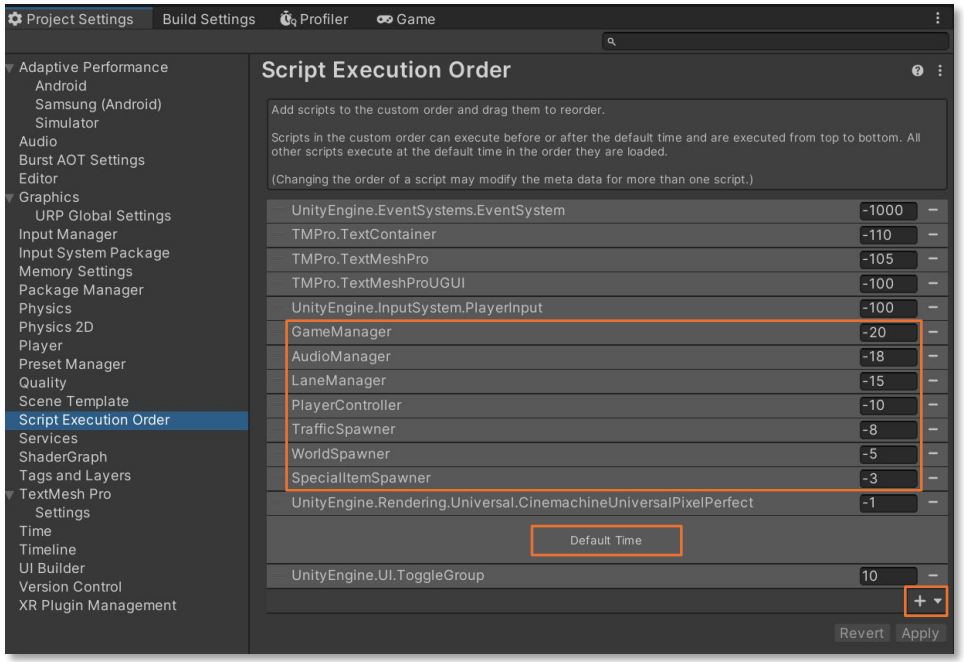
```
public class UsingClass : MonoBehaviour
{
  private void SomeFunc() {
    if (GameManager.instance != null) {
      GameManager.instance.SomePublicFunc();
    }
  }
}
```

```
public class NonPersistentClass : MonoBehaviour
{
  public static NonPersistentClass instance;

  private void Awake() {
    instance = this;
}
```

Observer

- One-to-many dependency between objects
- When subject changes state, observers react automatically and independently
- C# delegates are function pointers or references to functions
- Events are declared using delegates
- Subject declares and fires event, observers subscribe to event and receive notification when an event is fired
- Subject should be initialized before the observers
- Example: the score-keeping system should fire an event that the UI system observes, causing an update to the text display
    - Decouple logic from UI





Observer

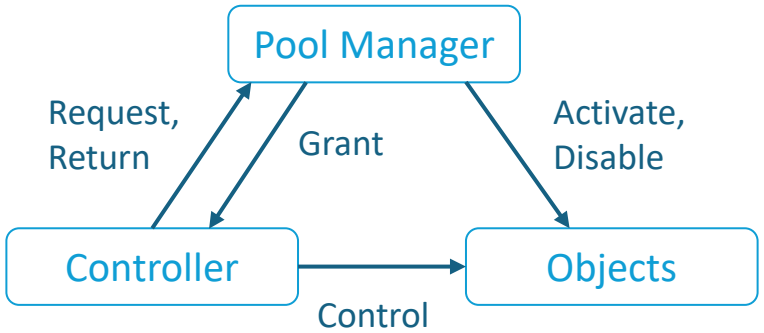
```
public Func<...> OnTimerTick;
public Action<...> OnTimerTick;

public event Action<...> OnTimerTick;
public event EventHandler<...> OnTimerTick;
```

```
public class GameManager : MonoBehaviour
{
  public delegate void OnTimerTick();
  public event OnTimerTick onTimerTick;

  float timerTick = 5f;
  float curTimerTick = 0f;

  private void Update() {
    curTimerTick += Time.deltaTime;
    if (curTimerTick >= timerTick) {
      curTimerTick = 0f;
      onTimerTick?.Invoke();
    }
  }
}
```

```
public class Enemy : MonoBehaviour
{
  private void OnEnable() {
    if (GameManager.instance != null) {
      GameManager.instance.onTimerTick += TimerTick;
    }
  }

  private void OnDisable() {
    if (GameManager.instance != null) {
      GameManager.instance.onTimerTick -= TimerTick;
    }
  }

  private void TimerTick() {
    // do something
  }
}
```

Object pooling

- Repeated create and destroy calls for game objects taxes the system, and invokes the garbage collector unnecessarily
- Create some quantity of an object during initialization, and keep them in deactivated state
- Request an object from this pool when needed, and activate it
- Return object to the pool when done, and deactivate it
- Implement for frequently used objects
    - Bullets in shooter games, obstacles in runner games
    - Particle system visual effects, sound effects



Object pooling

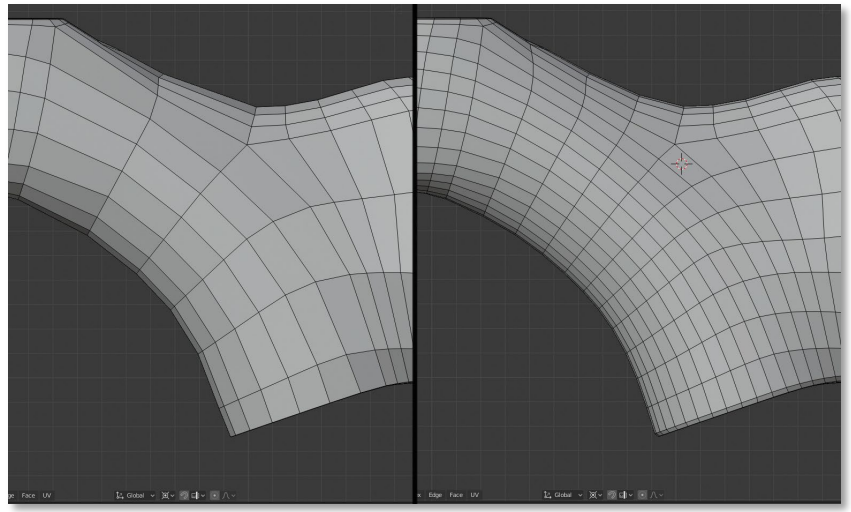
```
using UnityEngine.Pool;

public class TrafficSpawner : MonoBehaviour
{
  ObjectPool<GameObject> vehiclePool;
  bool collectionCheck = true; // throw error if we try to
                               release an item that is
                               already in the pool
  int defaultPoolSize = 30; int maxPoolSize = 50;
  // default size: pool memory we commit up front
  // max size: further allowance. If we return an item
     when the pool has hit this size, it is destroyed

  void Init() {
    vehiclePool = new ObjectPool<GameObject> (
                  CreatePooledVehicle,
                  GetPooledVehicle,
                  ReleasePooledVehicle,
                  DestroyPooledVehicle,
                  collectionCheck,
                  defaultPoolSize,
                  maxPoolSize);
  }
```

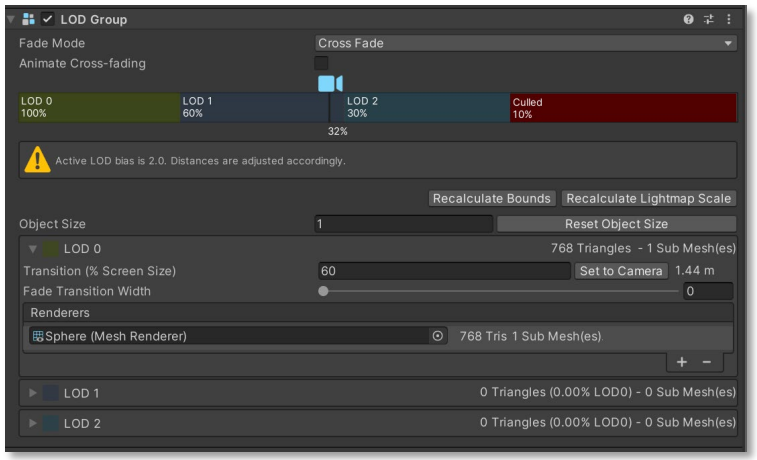
```
void DestroyPooledVehicle(GameObject vehicle) {
  Destroy(vehicle);
}

GameObject CreatePooledVehicle() {
  GameObject spawnedVehicle = Instantiate(...);
  return spawnedVehicle;
}

void GetPooledVehicle(GameObject vehicle) {
  // vehicle is available, perform desired actions
  vehicle.transform.position = respawnPoint;
}

void ReleasePooledVehicle(GameObject vehicle) {
  // perform actions to put away the vehicle
  vehicle.transform.position = parkPoint;
}
```

```
GameObject vehicle = vehiclePool.Get();

vehiclePool.Release(vehicle); // must retain reference
```

Models

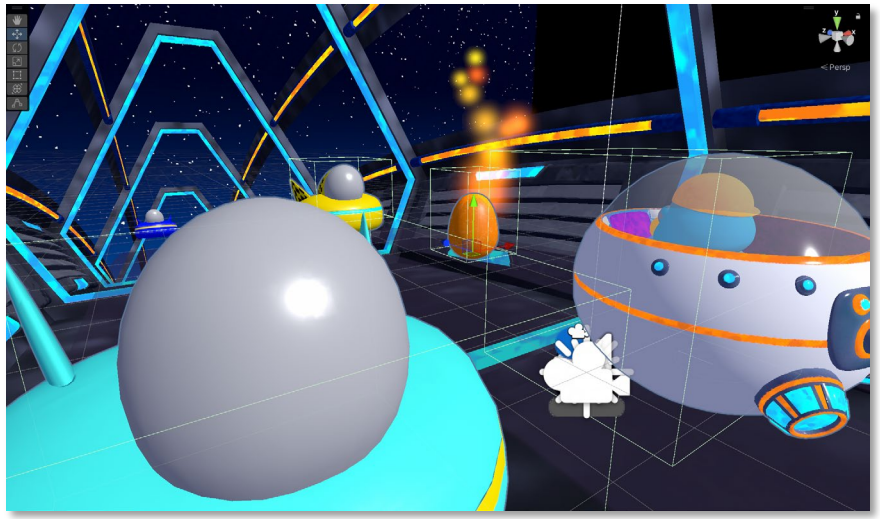
- Reduce 3D model polygon count
  - Memory usage, vertex data transfer to GPU
- Implement Level of Detail models

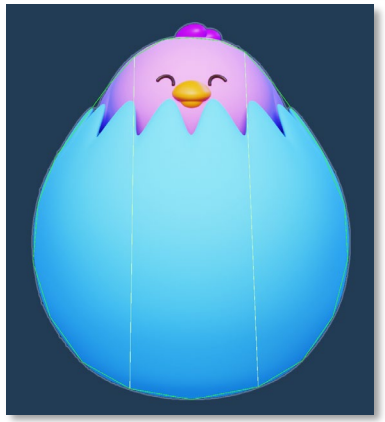




Models

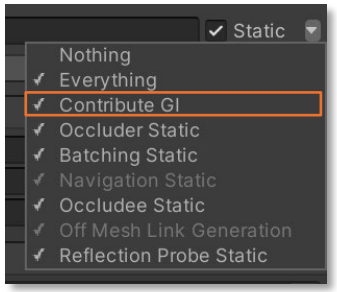
- Texture sizes should be appropriate for device and distance from camera
- Use the simplest collider possible
    - Set layer collision matrix in project settings to reduce checks

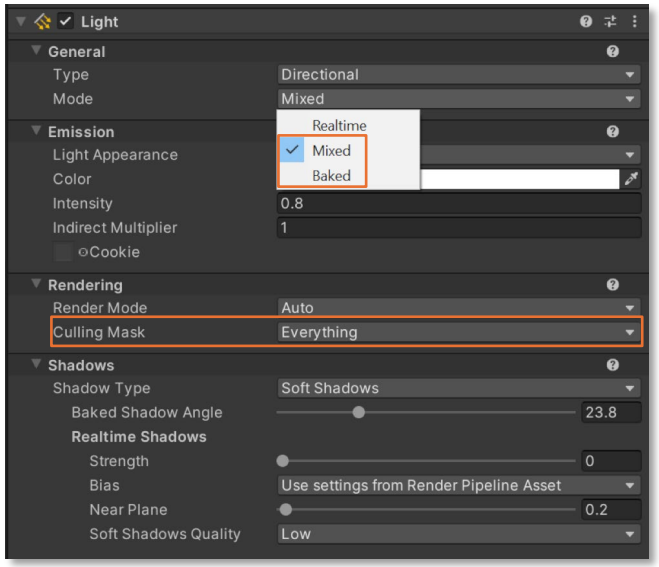


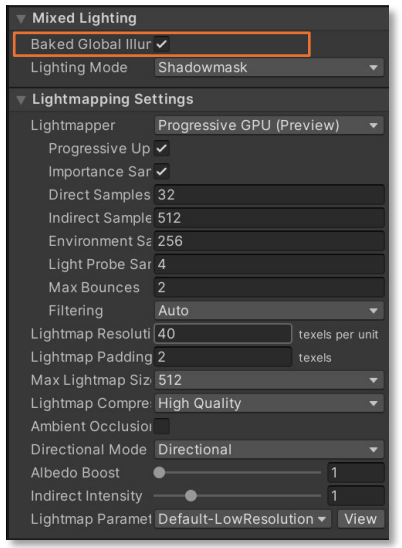


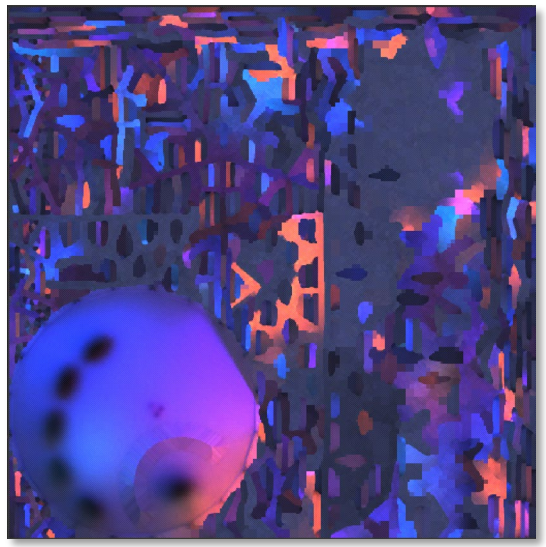
Graphics

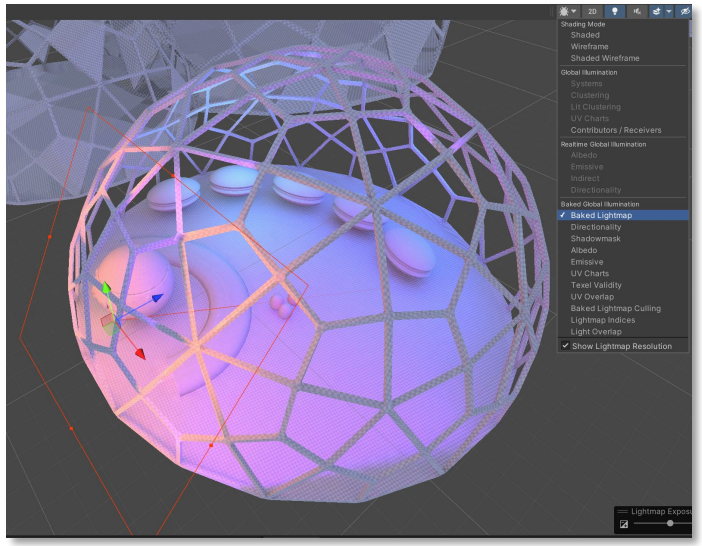
- Reduce real-time lights to limit expensive computations
- Baked lighting
  - Lightmaps: pre-compute and store lighting data in texture maps
  - Cannot be modified at runtime
  - Mark non-moving objects static for global illumination (Contribute GI)
  - Models must have “Generate lightmap UVs” enabled

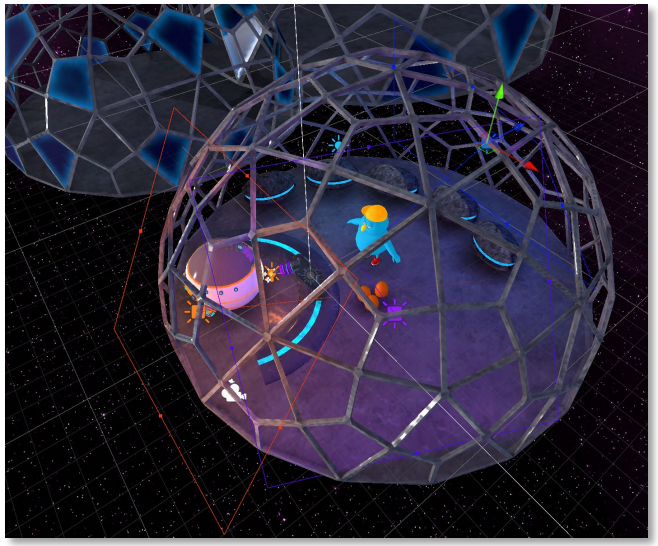






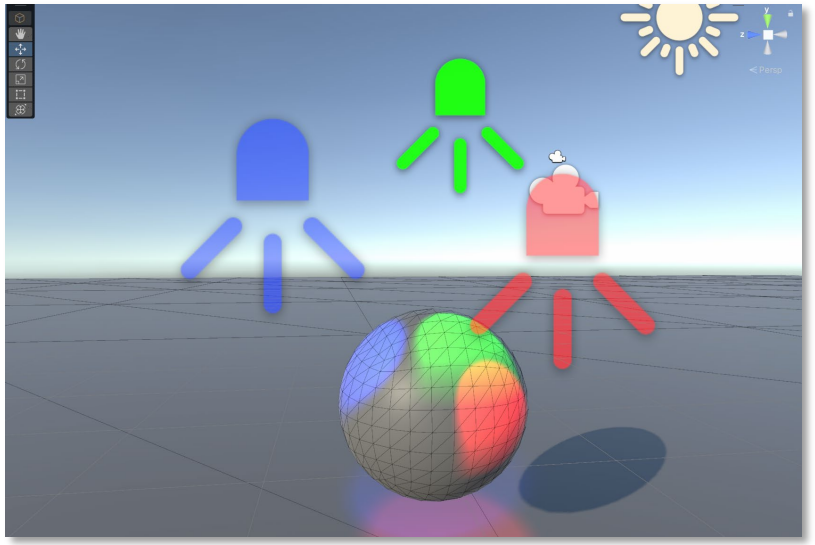


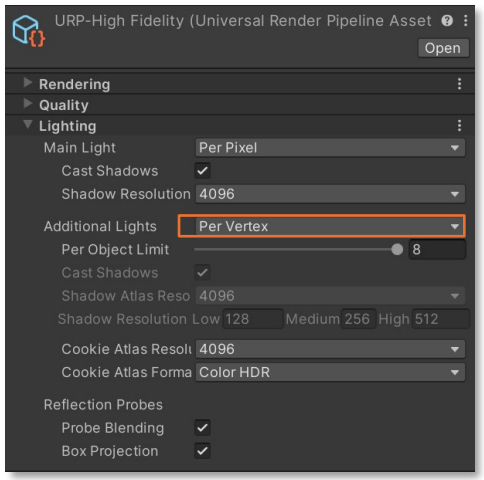


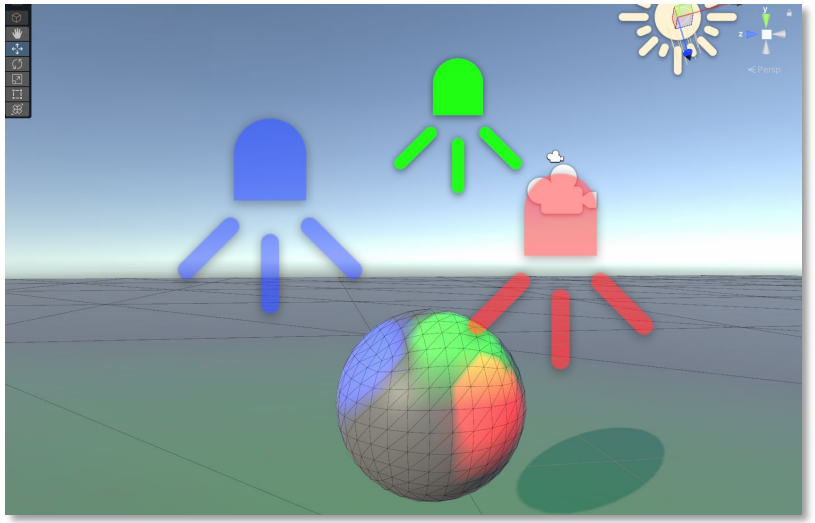


Graphics

- Per-pixel vs. per-vertex secondary lights
  - 3D model vertex count
  - Distance from camera

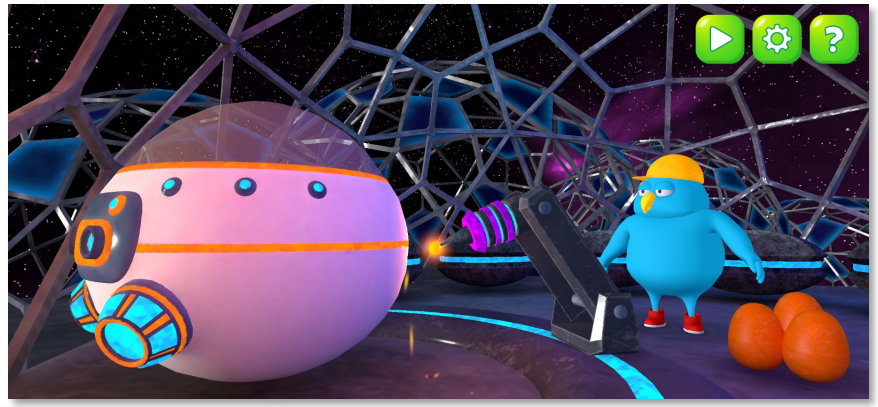


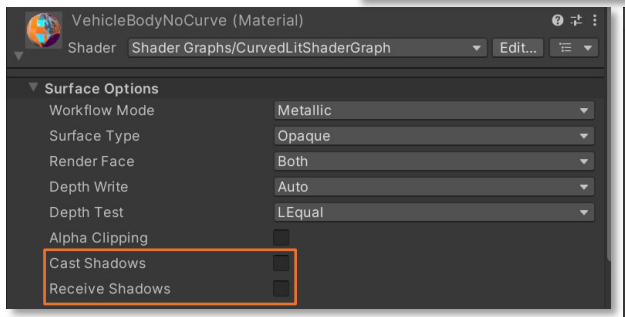


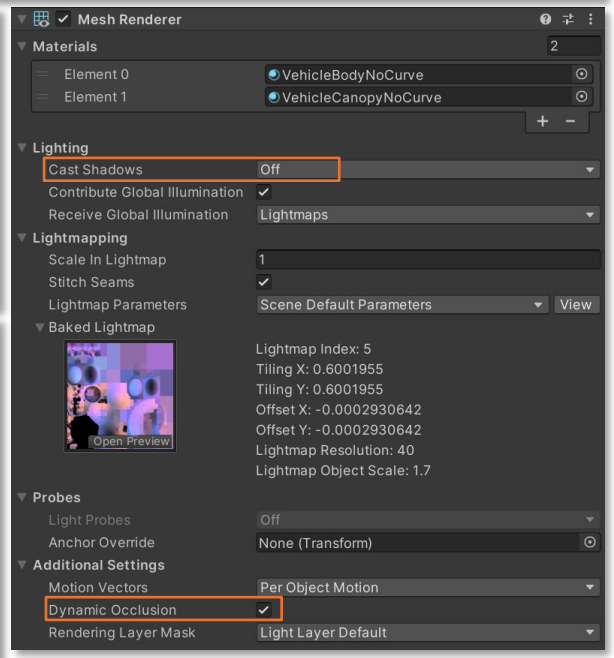


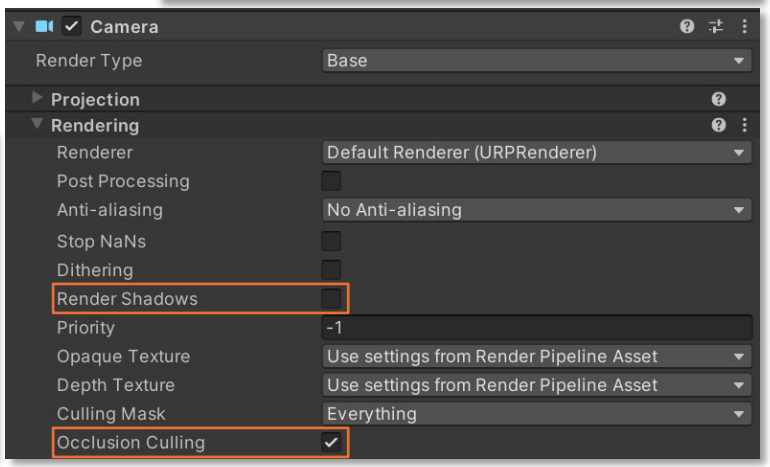
Graphics

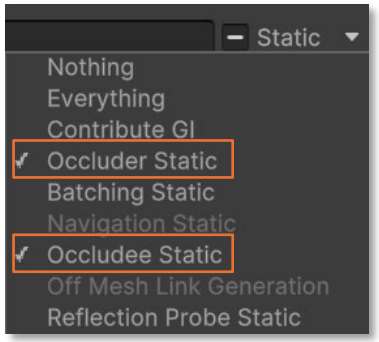
- Disable or reduce shadow quality
  - In mesh renderer, material properties, camera, and light settings
    - Camera settings override render pipeline settings
  - Fake shadow using 2D sprite
- Enable occlusion culling
  - Frustum culling happens automatically
  - Hide additional objects with occlusion culling
  - Dynamic
  - Static occluder: hide objects behind this one
  - Static occludee: hide this object if behind another
  - Bake information: Window > rendering > occlusion culling
  - Occlusion portal component





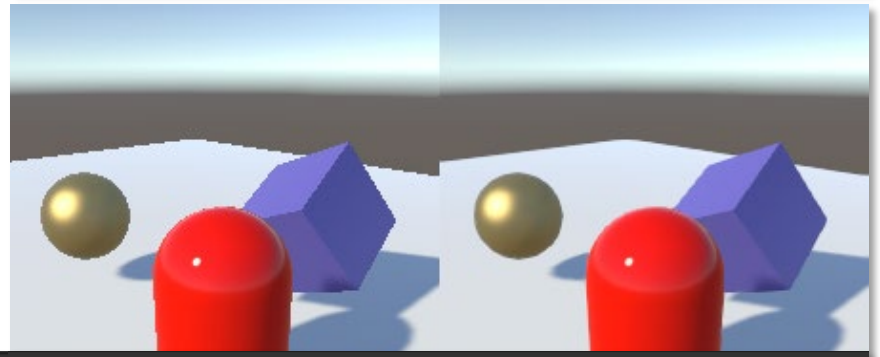


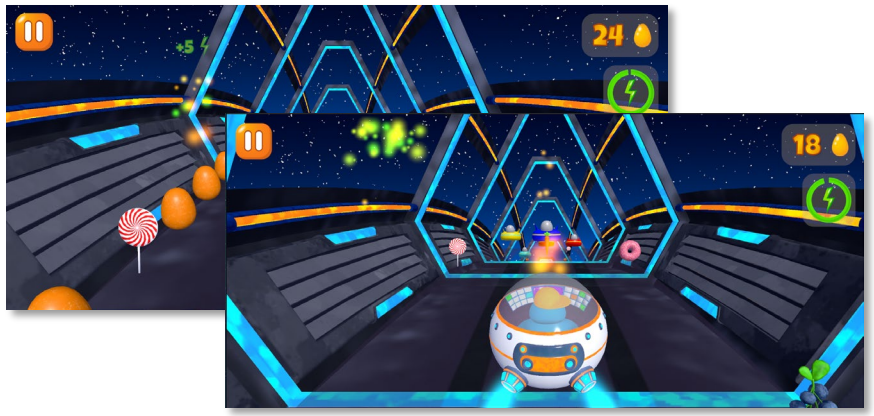


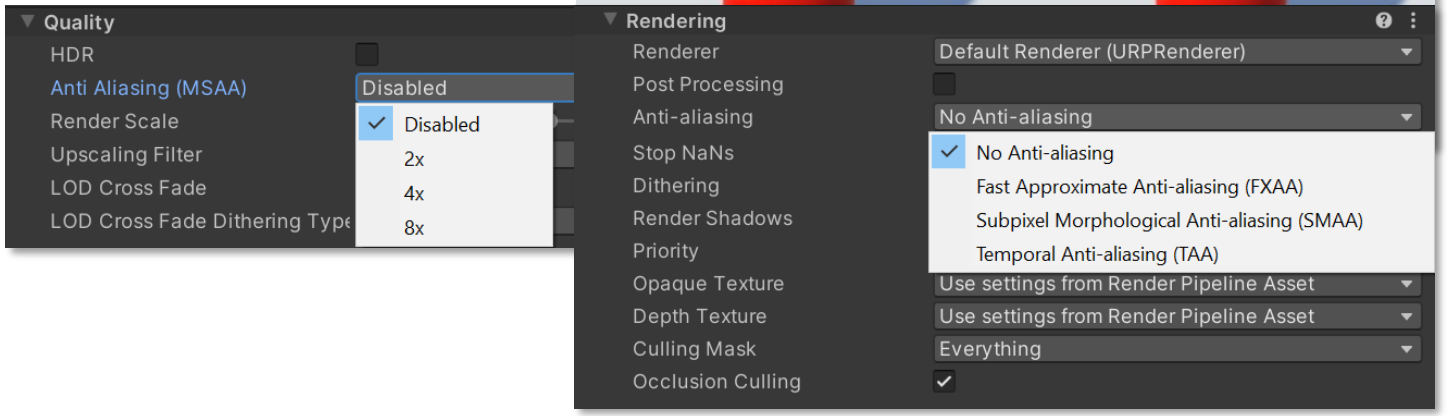


Graphics

- Remove post-processing effects such as bloom
- Reduce anti-aliasing quality
  - Render pipeline settings
  - Camera settings: rendering and MSAA
- Reduce VFX particle count





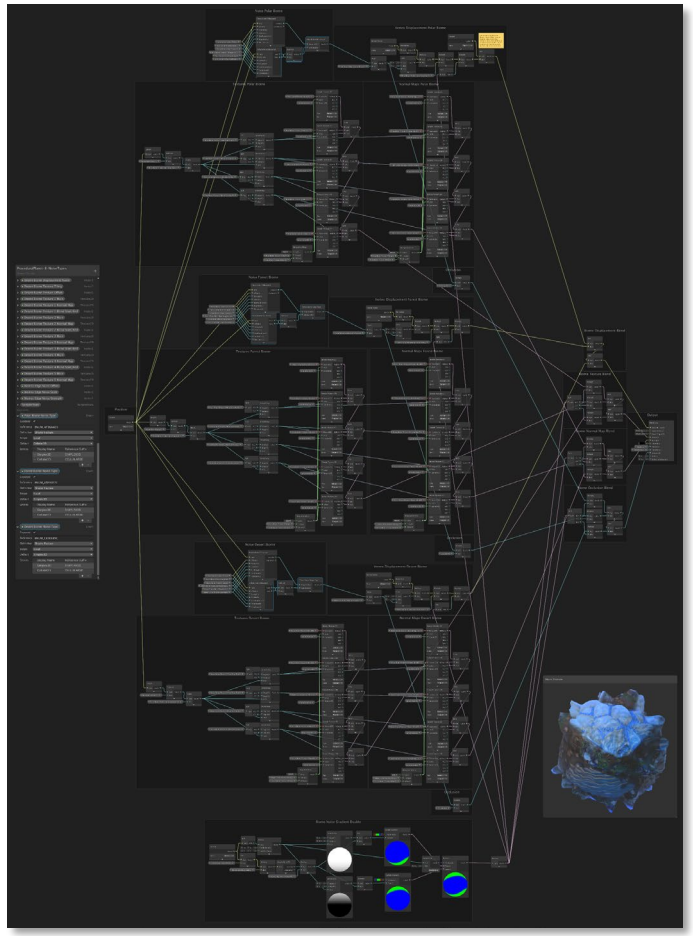


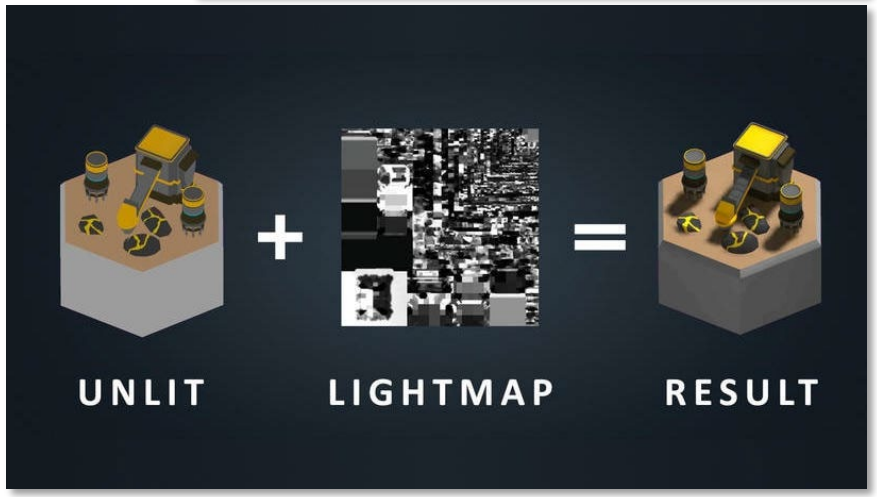
Graphics

- All elements on a canvas get refreshed if a single element changes
- Have separate UI canvases for each UI screen
- Disable raycast target for UI elements that the player does not interact with

Graphics

- Shaders
  - Programs that run on GPU
  - Perform calculations to determine color and position of pixels
- Simplify shaders
  - Avoid unnecessary features
  - Reduce computation precision (float, half)
  - Reduce usage of exp, cos, log functions
- Unlit shaders have fewer calculations
  - Work well with stylized art
  - Use in conjunction with lightmaps



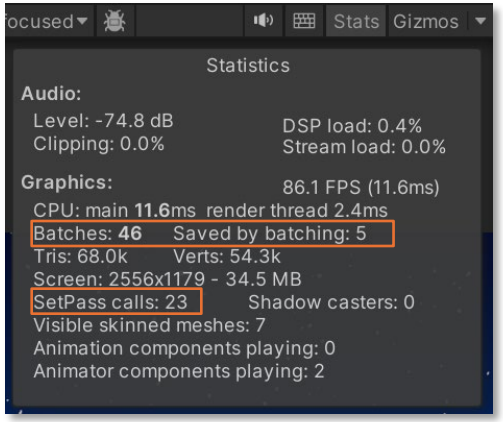


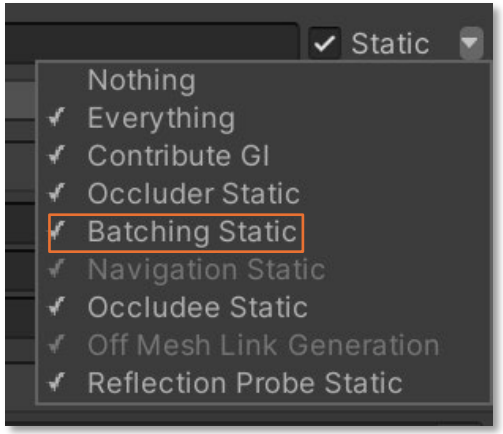
Graphics

- Draw calls
  - Processed by CPU first, then transferred to GPU to draw geometry on the screen
  - Contains information related to mesh, textures, materials, shaders

Meshes	Materials	Draw calls *
1	1	1
1	2	2
2	1	2
2	2	4

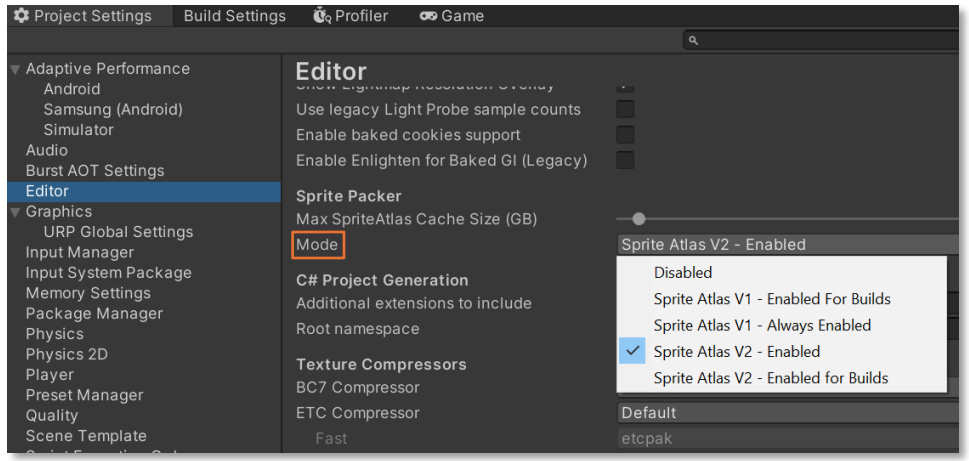
  - Reduce by
    - Combining meshes in the 3D model
    - Submitting multiple objects as a single object for drawing
      - Static draw call batching: mark stationary objects static so their meshes are combined, must have the same material
      - GPU instancing: non-static objects (moveable), same mesh, same material
- Shader pass: collection of settings and functions for shader configuration, i.e. material setup changes
  - Each pass renders object differently
  - E.g. toon shader: one pass does the effect, second pass draws outline
- SetPass calls
  - Number of times shader pass is switched during a frame
  - Reduce by reusing materials and limiting shader complexity





Graphics

- Draw call issued for each texture in the scene
- Sprite atlas
  - Unified texture created from individual sprites
  - Single draw call to access all packed sprites
  - All UI elements in one UI canvas should be packed into a sprite atlas to reduce draw calls
  - Enable in project settings



DOTS

- Data-Oriented Technology Stack
  - Over 200x speed-up in some scenarios
- Entity Component System
  - Structs, not classes
    - Contiguous memory locations reduce cache misses in hardware
    - Caution: structs are value types, not references types like classes
  - Entity: identifier
  - Component: data holder. IComponentData
  - System: logic. ISystem
- Job system
  - Multi-threading with safety checks
- Burst compiler
  - Data-oriented code optimizations