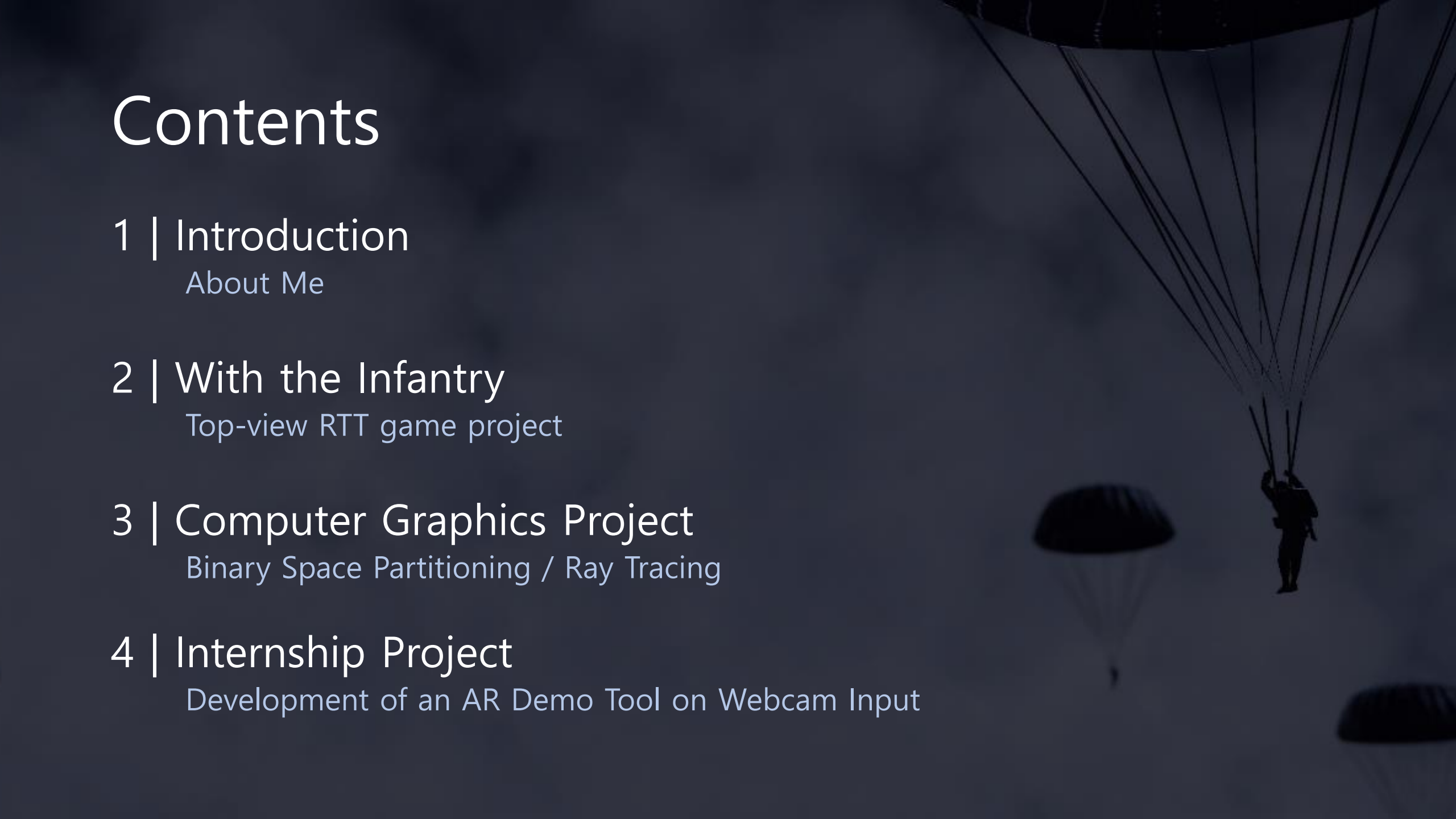
PORTFOLIO

Park Changhwi

smsychjy96@gmail.com

https://github.com/ObjectOrientedLife

Contents



1 | Introduction

About Me

Introduction About Me

- Changhwi Park
- smsychjy96@gmail.com
- https://github.com/ObjectOrientedLife
- Education - Seoul Nation University, geography / computer Science
- Interests - Game development / computer graphics(realtime rendering, shader, ray tracing, humanoid animation) / computer vision / human pose estimation
- Languages - C#, C++, Python, Shader language(Cg)
- Skills - Unity, OpenGL, OpenCV / Blender, ZBrush, Substance Painter

2 | With the Infantry

Top-view RTT game project

1) Roles
2) Programming
3) Art
4) Demo

1) Roles - One-man Development

Took charge of every stage of the game development

Design

- Gameplay concept
- Planning
- Background research

Art

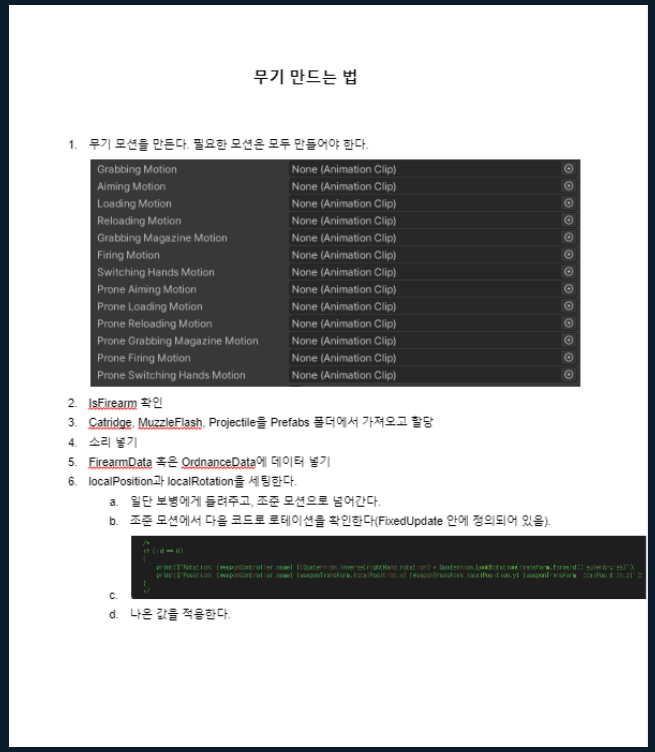
- Modeling
- Sculpting / texturing
- Rigging / skinning / animating

Programming

- Unity
- Client(C#)
- Server(Unity Mirror)
- Shader(Cg)

1) Roles After Building a Team

- Organized a team composed of three members
- Communication - hosting meetings, scheduling, feedback
- Establishing modeling processes
- Writing design specifications
- Tutoring how to utilize modeling tools



무기 만드는 법

1. 무기 모션을 만든다. 필요한 모션은 모두 만들어야 한다.

Grabbing Motion	None (Animation Clip)
Aiming Motion	None (Animation Clip)
Loading Motion	None (Animation Clip)
Reloading Motion	None (Animation Clip)
Grabbing Magazine Motion	None (Animation Clip)
Firing Motion	None (Animation Clip)
Switching Hands Motion	None (Animation Clip)
Prone Aiming Motion	None (Animation Clip)
Prone Loading Motion	None (Animation Clip)
Prone Reloading Motion	None (Animation Clip)
Prone Grabbing Magazine Motion	None (Animation Clip)
Prone Firing Motion	None (Animation Clip)
Prone Switching Hands Motion	None (Animation Clip)

2. IsFirearm 확인
3. Catridge, MuzzleFlash, Projectile을 Prefabs 폴더에서 가져오고 할당
4. 소리 넣기
5. FirearmData 혹은 OrdnanceData에 데이터 넣기
6. localPosition과 localRotation을 세팅한다.
   a. 일단 보병에게 들려주고, 조준 모션으로 넘어간다.
   b. 조준 모션에서 다음 코드로 로테이션을 확인한다(FixedUpdate 안에 정의되어 있음).

```
/*
if (id == 0)
{
    print("Rotation (weaponController.name) (Quaternion.Inverse(rightHand.rotation) * Quaternion.LookRotation(transform.forward)).eulerAngles)");
    print("Position (weaponController.name) (weaponTransform.localPosition.x) (weaponTransform.localPosition.y) (weaponTransform.localPosition.z)");
}
*/
```

   c.
   d. 나온 값을 적용한다.

2) Programming Client

Hardships

- Codes becoming too verbose due to a variety of weapons(single action, automatic) and vehicles(howitzers, tanks, aircrafts)
- Hard to understand interactions between modules

- Requirement for optimization
- Readability - optimization tradeoff
- Memory - time tradeoff(ex - object pooling)

- Limited modeling capability
- Hard to maintain coherency

Solutions

- Inheritance structure based on the SOLID principles that enhances reusability
- Separation of interfaces and implementations using design patterns(singleton, abstract factory, strategy)

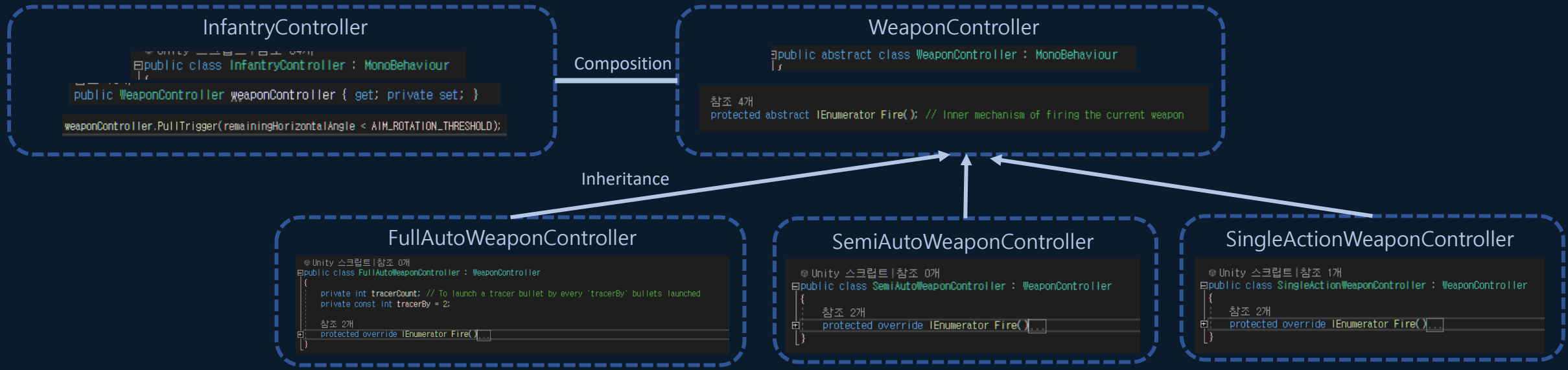
- Classify game stages into four hierarchies(initialization - occasionally during a playtime - frequently during a playtime - every frame) and apply different levels of optimization

- Rescues to the reusability of the resources
- Customizable character

2) Programming Strategy Pattern - Weapon

- Every soldier owns a firearm
- Weapons feature several common functionalities, including launching bullets, ejecting cartridges, and reloading
- Make a WeaponController class that embraces all the weapons

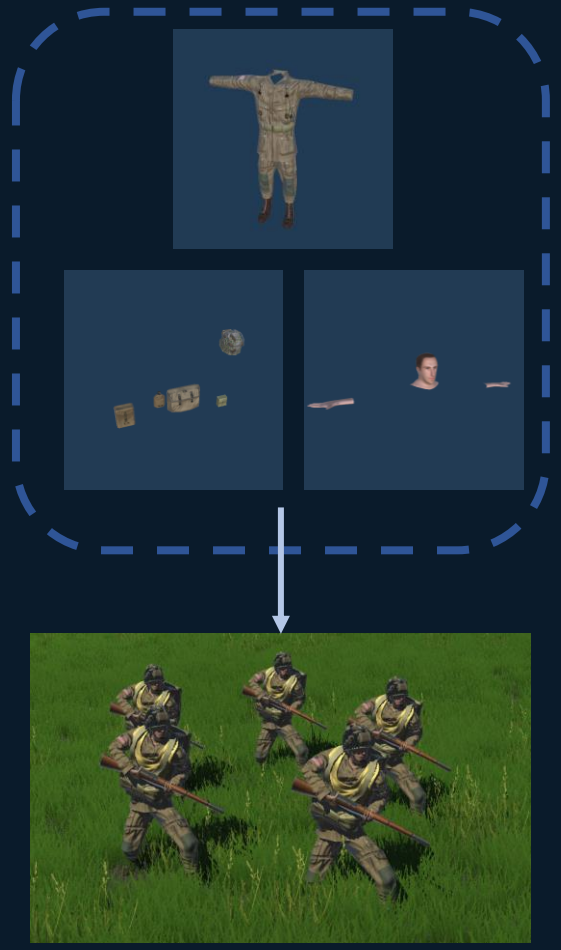
- Classify weapons into full-auto / semi-auto / single-action, and their classes inherits from the WeaponController class
- A soldier can fire his weapon by just calling weaponController.PullTrigger(), without taking care of the actual mechanism
- Separation of the interface and the implementation



2) Programming Character Customizer

- Too many combinations of clothing / bodies / gadgets
- Produce clothing / bodies / gadgets separately and combine them together in Unity
- Replace the deformable object itself and the bones
- Undeformable gadgets are managed through tags

```
2 references
public void SetMesh(Child child, string customName)
{
    GameObject childToChange;
    GameObject target;

    if (child == Child.body)
    {
        childToChange = body;
        target = CommonResources.commonResources.bodies[customName];
    }
    else
    {
        childToChange = clothing;
        target = CommonResources.commonResources.clothing[customName];
    }

    // Instantiate a prefab and set as a child of the parent
    GameObject targetToChange = Instantiate(target);
    targetToChange.transform.SetParent(transform);
    targetToChange.transform.localPosition = Vector3.zero;
    targetToChange.name = target.name;

    ChangeBone(childToChange, targetToChange);
    if (child == Child.clothing)
    {
        ChangeGadget(targetToChange);
    }

    Destroy(childToChange.gameObject);
    Destroy(targetToChange.transform.Find("Infantryman").gameObject); // Destroy bones of the target object
}

1 reference
private void ChangeBone(GameObject childToChange, GameObject targetToChange)
{
    SkinnedMeshRenderer thisRenderer = childToChange.GetComponent<SkinnedMeshRenderer>();
    SkinnedMeshRenderer targetRenderer = targetToChange.GetComponent<SkinnedMeshRenderer>();
    Dictionary<string, Transform> boneMap = new Dictionary<string, Transform>();

    Transform[] boneArray = targetRenderer.bones;

    foreach (Transform bone in thisRenderer.bones)
    {
        boneMap[bone.name] = bone;
    }

    // Check if the bones match
    for (int i = 0; i < boneArray.Length; i++)
    {
        string boneName = boneArray[i].name;

        if (boneMap.TryGetValue(boneName, out boneArray[i]) == false)
        {
            Debug.LogError("failed to get bone: " + boneName);
        }
    }

    targetRenderer.bones = boneArray; // Take effect
}
```



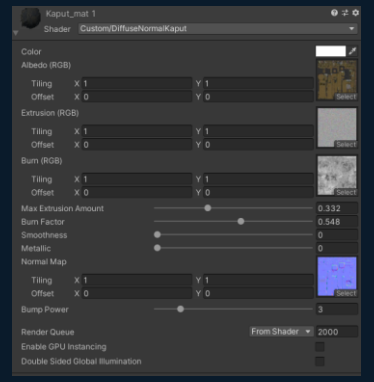
2) Programming Animation State Callback

- Unity provides ways to detect animation state changes with the StateMachineBehaviour
- StateMachineBehaviour is hard to be modified during a runtime
- Wrote a trick script to resolve the limitation
- By hiring the callback pattern, the performance gets better and the code becomes more succinct
- Used for weapon reloading motions
  - Set a magazine as a child of a hand upon the reloading motion starts
  - Set the magazine as a child of the firearm upon the reloading motion ends

```
public enum CallbackType { Enter, Exit, Update, Move, IK }
Unity Script | 3 references
public class StateEvent : StateMachineBehaviour
{
    private Dictionary<int, Dictionary<CallbackType, UnityEvent>> callbacks = new Dictionary<int, Dictionary<CallbackType, UnityEvent>>();

    1 reference
    public void AddCallback(CallbackType callbackType, string tag, UnityAction callback)
    {
        int tagHash = Animator.StringToHash(tag);
        if (!callbacks.ContainsKey(tagHash))
        {
            callbacks[tagHash] = new Dictionary<CallbackType, UnityEvent>();
        }

        if (!callbacks[tagHash].ContainsKey(callbackType))
        {
            callbacks[tagHash][callbackType] = new UnityEvent();
        }

        callbacks[tagHash][callbackType].AddListener(callback);
    }

    0 references
    public void ClearCallbacks(CallbackType callbackType, string tag)
    {
        int tagHash = Animator.StringToHash(tag);
        if (callbacks.ContainsKey(tagHash) && callbacks[tagHash].ContainsKey(callbackType))
        {
            callbacks[tagHash][callbackType].RemoveAllListeners();
        }
    }

    1 reference
    public void ClearCallbacks(string tag)
    {
        int tagHash = Animator.StringToHash(tag);
        if (callbacks.ContainsKey(tagHash))
        {
            foreach (var kv in callbacks[tagHash])
            {
                kv.Value.RemoveAllListeners();
            }
        }
    }

    Unity Message | 0 references
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        if (callbacks.ContainsKey(stateInfo.tagHash) && callbacks[stateInfo.tagHash].ContainsKey(CallbackType.Enter))
        {
            callbacks[stateInfo.tagHash][CallbackType.Enter].Invoke();
        }
    }

    Unity Message | 0 references
    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        if (callbacks.ContainsKey(stateInfo.tagHash) && callbacks[stateInfo.tagHash].ContainsKey(CallbackType.Exit))
        {
            callbacks[stateInfo.tagHash][CallbackType.Exit].Invoke();
        }
    }
```

2) Programming Shader

- Cg shader language
- Surface shader that simulates a destruction effect without making new models
  - Vertex modifier extrudes vertices according to the 'noise texture'
  - Surface function overlays the 'burn texture'
- Several parameters enable a variety of different destruction effects



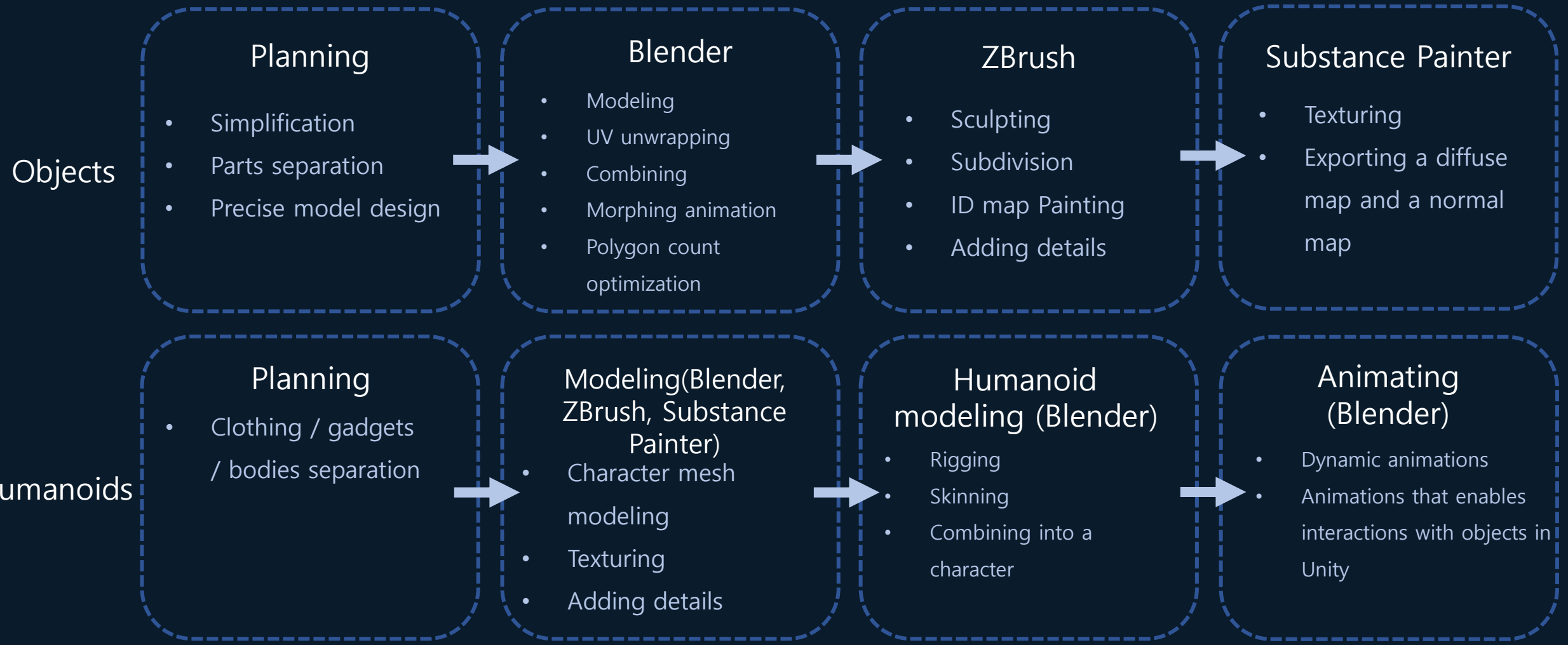

```
void vert(inout appdata_full v)
{
    float4 tex = tex2Dlod(_ExtrusionTex, float4(v.vertex.xyz, 0));
    tex = tex * 2 - 1;
    v.vertex.xyz += _ExtrusionAmount * tex.rgb;
}
```

```
void surf(Input IN, inout SurfaceOutputStandard o)
{
    fixed4 c = tex2D(_MainTex, IN.uv_MainTex) * _Color;
    fixed4 burn = tex2D(_BurnTex, IN.uv_MainTex);

    o.Albedo = c.rgb * burn.r * _BurnFactor;
    o.Metallic = _Metallic;
    o.Smoothness = _Glossiness;
    o.Alpha = c.a;

    fixed3 normal = UnpackNormal(tex2D(_BumpMap, IN.uv_BumpMap));
    normal.z = normal.z / _BumpPower;
    o.Normal = normalize(normal);
}
```

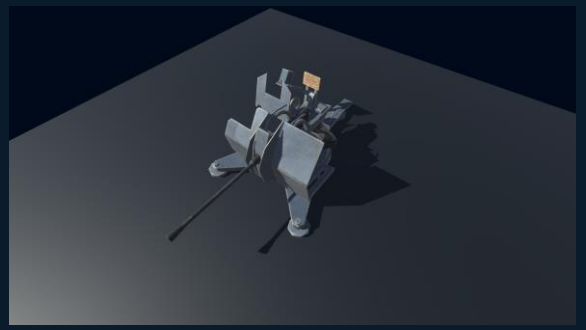
Original model

Destructed model

3) Art Objects / Humanoid Modeling



3) Art Works



3) Demo Trailer

- https://www.youtube.com/watch?v=b9b-6MzOAi0
- https://www.youtube.com/watch?v=zgcS1foEgOA

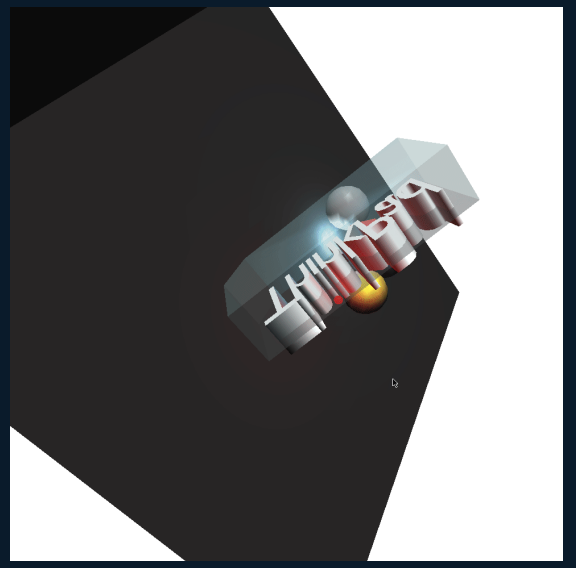
3 | Computer Graphics Project

Binary Space Partitioning / Ray Tracing

Computer Graphics Project Binary Space Partitioning

- C++ / OpenGL scene
- Default OpenGL depth test is erroneous when translucent and opaque objects are placed altogether

No BSP - objects behind are hidden by the translucent object

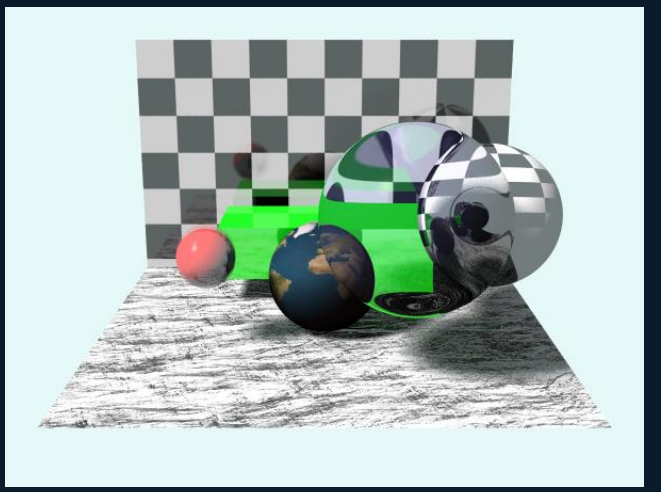


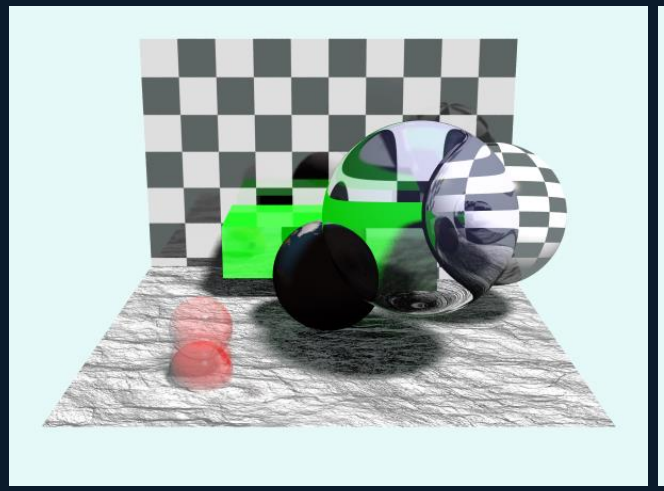
BSP - objects behind are rendered correctly without being hidden

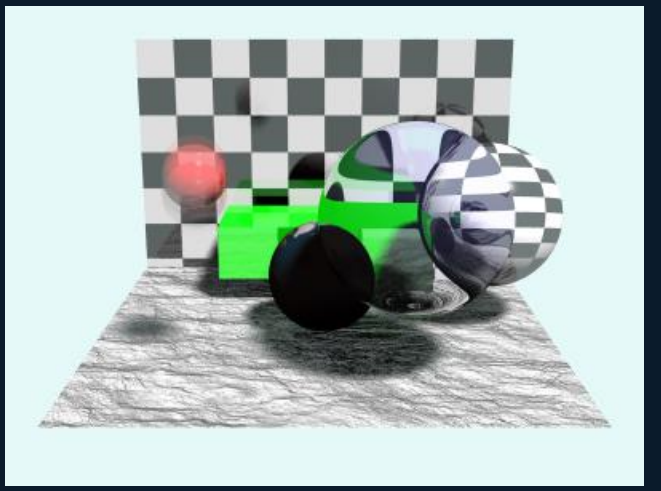
- Binary Space Partitioning(BSP) to the rescue!
- Build a BSP tree once, traverse every frame
- Traversing the BSP tree correctly conducts a depth test regardless of the viewpoint
- Source code is available on https://github.com/ObjectOrientedLife/BinarySpacePartitioning

Computer Graphics Project Ray Tracing

- Implemented with pure C++(no OpenGL)
- Whitted Ray tracing based on the Phong illumination - recursive refraction, recursive refraction
- Stochastic ray tracing through subpixel sampling - soft shadow, motion blur
- Supports textures and normal maps
- Supports .obj parser
- Multithreading(x3 faster on a 8-core CPU)
- Source code is available on https://github.com/ObjectOrientedLife/RayTracing







Computer Graphics Project Ray Tracing

1. Modularized into Scene, Light, Shape(Face, Sphere)

```
class Scene
{
public:
    void insertFaces(vector<Face> object, mat4x4 transformation, vec3 diffuse, vec3 specular, vec3 emission, float shininess, float alpha, float refractionFactor);
    void insertSphere(Sphere sphere, mat4x4 transformation, vec3 diffuse, vec3 specular, vec3 emission, float shininess, float alpha, float refractionFactor);
    void insertLight(Light light);
    void insertMovementSphere(vector<vec3> movement);
    Hit getClosest(Ray ray);
    vec3 Phong(vec3 hitPos, vec3 N, vec3 V, vec3 diffuse, vec3 specular, float shininess); // pixel position, normal of that pixel, where the ray comes from, material properties
    void move(int sampleIdx);
    void restore();

private:
    vector<Face> faces;
    vector<Sphere> spheres;
    vector<Light> lights;
    map<int, vector<vec3>> sphereMovements;
    vector<Sphere> spheresBackup;
};

struct Ray
{
    Ray(vec3 _start, vec3 _dir): start(_start), dir(_dir) {}
    vec3 start;
    vec3 dir; // Unit vector that represents the direction
};

struct Hit
{
    shared_ptr<Shape> shape;
    vec3 pos;
    float dist = INFINITY; // Distance that the ray travelled
};

#endif
```

Scene.h

Composition

Shape.h

```
struct Shape
{
    Shape() {}
    Shape (vec4 _diffuse, vec4 _specular, vec4 _emission, float _shininess, float _alpha, float _refractionFactor): diffuse(_diffuse), specular(_specular), emission(_emission), shininess(_shininess), alpha(_alpha), refractionFactor(_refractionFactor)  {}
    virtual ~Shape() {}

    // Members
    bool isSphere = false;

    vec3 diffuse;
    vec3 specular;
    vec3 emission;
    float shininess;
    float alpha;
    float refractionFactor;

    mat4x4 transform;

    // UV coordinate data
    bool hasTexture = false;
    vector<vec3> *texture;
    int textureW;
    int textureH;
};
```

Inheritance

Face.h, Sphere.h

```
struct Light
{
    Light(vec3 _pos, vec3 _aimAt, vec3 _diffuse, float _quadAtten, bool _isFar, float _areaFactor): pos(_pos), aimAt(_aimAt), diffuse(_diffuse), quadAtten(_quadAtten), isFar(_isFar), areaFactor(_areaFactor) {}

    vec3 pos;
    vec3 aimAt;
    vec3 diffuse;
    float quadAtten;
    bool isFar;
    float areaFactor;
};
```

Light.h

Computer Graphics Project Ray Tracing

main.cpp

2. Load texture and .obj files

```
void initObjects()
{
    // ==================== Initalize objects ====================
    led = Sphere(vec3(0, 0, 0), 0.1, vec3(0, 0, 0));
    thinkPad = parseData("./Models/ThinkPad.obj");
    panel = parseData("./Models/Panel.obj");
    key = parseData("./Models/Key.obj");
    trackPoint = parseData("./Models/TrackPoint.obj");
    cube = parseData("./Models/Cube.obj");
    sphere = Sphere(vec3(0, 0, 0), 0.5, vec3(0, 0, 0));
    goldSphere = Sphere(vec3(0, 0, 0), 0.5, vec3(0, 0, 0));
    largeSphere = Sphere(vec3(0, 0, 0), 1.5, vec3(0, 0, 0));
    checkerSphere = Sphere(vec3(0, 0, 0), 1.3, vec3(0, 0, 0));
    earth = Sphere(vec3(0, 0, 0), 1, vec3(0, 0, 0));
    plane = parseData("./Models/Plane.obj");
    checker = parseData("./Models/Plane.obj");
    rock = parseData("./Models/Plane.obj");

    // ==================== Initalize textures ====================
    readBMP("./Textures/Carbon.bmp", &carbonTexture, &carbonW, &carbonH);
    plane[0].setTexture(&carbonTexture, carbonW, carbonH, vec2(carbonW - 1, carbonH - 1), vec2(0, 0), vec2(0, carbonH - 1));
    plane[1].setTexture(&carbonTexture, carbonW, carbonH, vec2(carbonW - 1, carbonH - 1), vec2(carbonH - 1, 0), vec2(0, 0));

    readBMP("./Textures/CarbonNormal.bmp", &carbonNormal, &carbonNormalW, &carbonNormalH);
    plane[0].setNormalMap(&carbonNormal, carbonNormalW, carbonNormalH, vec2(carbonNormalW - 1, carbonNormalH - 1), vec2(0, 0), vec2(0, carbonNormalH - 1));
    plane[1].setNormalMap(&carbonNormal, carbonNormalW, carbonNormalH, vec2(carbonNormalW - 1, carbonNormalH - 1), vec2(carbonNormalH - 1, 0), vec2(0, 0));
```

3. Compose a scene from each module

```
// ==================== Place objects onto the scene ====================
mat4x4 identity = mat4x4(1.0f);
mat4x4 transformScene = rotate(identity, 15.0f * (float) M_PI / 180.0f, vec3(1, 0, 0)); // Transforms the whole scene
transformScene = rotate(transformScene, 15.0f * (float) M_PI / 180.0f, vec3(0, 1, 0));

mat4x4 transformPlane = translate(transformScene, vec3(0, 0, 0));
scene.insertFaces(plane, transformPlane, vec3(0.1, 0.1, 0.1), vec3(0.01, 0.01, 0.01), vec3(0, 0, 0), 1, 1, 1);

mat4x4 transformBackground = translate(transformScene, vec3(0, 0, -4));
transformBackground = rotate(transformBackground, 90.0f * (float) M_PI / 180.0f, vec3(1, 0, 0));
scene.insertFaces(plane, transformBackground, vec3(0.1, 0.1, 0.1), vec3(0.01, 0.01, 0.01), vec3(0, 0, 0), 1, 1, 1);

mat4x4 transformGoldenSphere = translate(transformScene, vec3(1.2, 0.5, 1.5));
scene.insertSphere(goldSphere, transformGoldenSphere, vec3(0.88, 0.75, 0.3), vec3(1, 0.84, 0), vec3(0, 0, 0), 7, 1, 1);
```

4. Conduct ray tracing

```
// ==================== Ray tracing ====================
vector<vec3> sampleResults[SAMPLE_COUNT];
#pragma omp parallel for // Execute in parallel
for (int s = 0; s < SAMPLE_COUNT; ++s) // s: Sample
{
    vector<vec3> imgVector;
    vec3 origin(0, 0, F);
    scene.move(s); // Move the scene
    for (int y = 0 ; y < H; ++y)
    {
        cout << "Sample: " << s << ", tracing y: " << y << "/" << H - 1 << endl;
        for (int x = 0; x < W; ++x)
        {
            float offsetX = getRandomFloat(0, 1);
            float offsetY = getRandomFloat(0, 1);

            float xPos = (-W / 2 + x + offsetX) * COEF; // Center pixel at the origin
            float yPos = (H / 2 - y + offsetY) * COEF;
            vec3 pixelPos(xPos, yPos, 0);

            vec3 primaryRay = normalize(pixelPos - origin);

            vec3 traced = traceRay(origin, primaryRay, vec3(0, 0, 0), 1, AIR_FACTOR);
            imgVector.push_back(traced);
        }
    }
    scene.restore(); // Restore the scene
    sampleResults[s] = imgVector;
}
```

```
vec3 traceRay(vec3 from, vec3 dir, vec3 color, int depth, float prevRefractionFactor)
{
    Ray ray(from, dir);
    Hit hit = scene.getClosest(ray);

    if (hit.dist != INFINITY) // If the ray hits something
    {
```

5. Export the result

```
vector<vector<vec3>> samples;
for (int i = 0; i < SAMPLE_COUNT; ++i)
{
    samples.push_back(sampleResults[i]);
}
vector<vec3> averaged = getAverageImage(samples); // Take the average

writeBMP("./Results/Result.bmp", W, H, averaged);
```

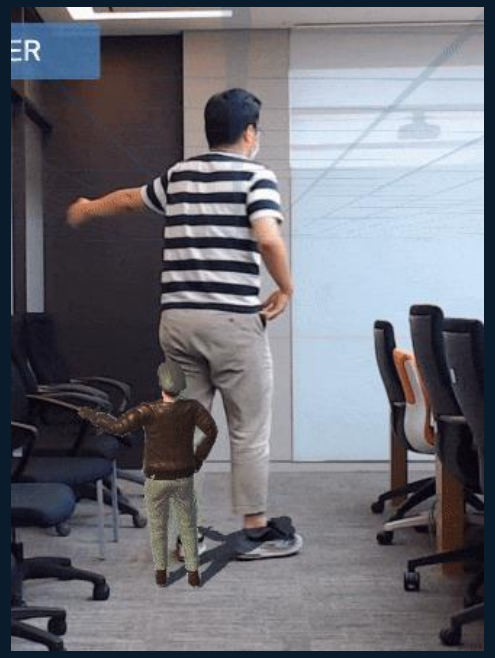
4 | Internship project

Development of an AR Demo Tool on Webcam Input

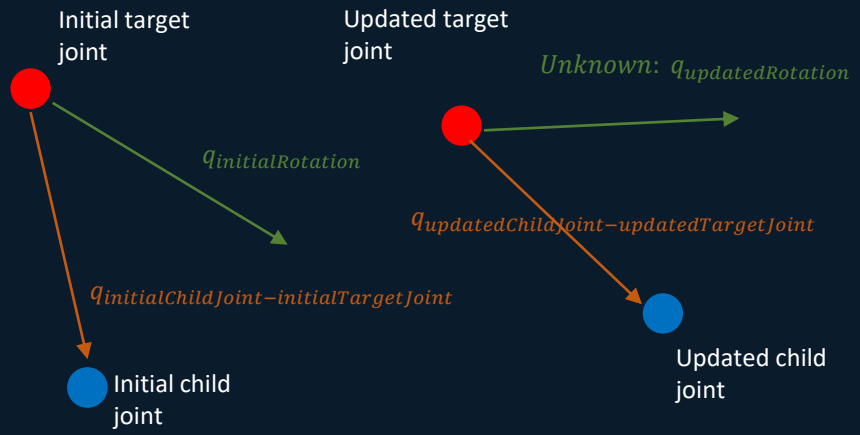
Internship project

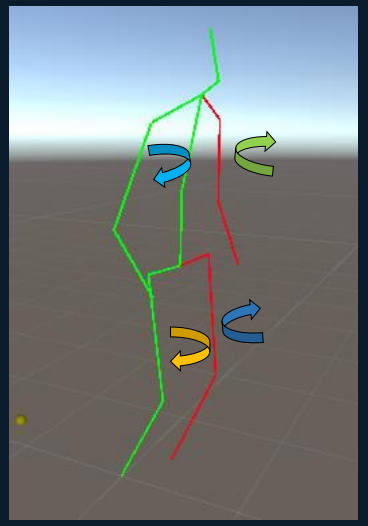
Development of an AR Demo Tool on Webcam Input

- NCSoft / Vision AI Lab / Human Pose Estimation team
- Conducted for two months(2021.07 - 2021.08)
- Feed webcam input into neural networks and present the results with a humanoid
- Kalman filter and low pass filter for smoothing motions
- Vanishing point determination / detection for deciding an AR ground
- Shadow shader that supports the AR component



Internship project Character Representation





$$q_{initialChildJoint-initialTargetJoint} \cdot q_{diff} = q_{initialRotation}$$

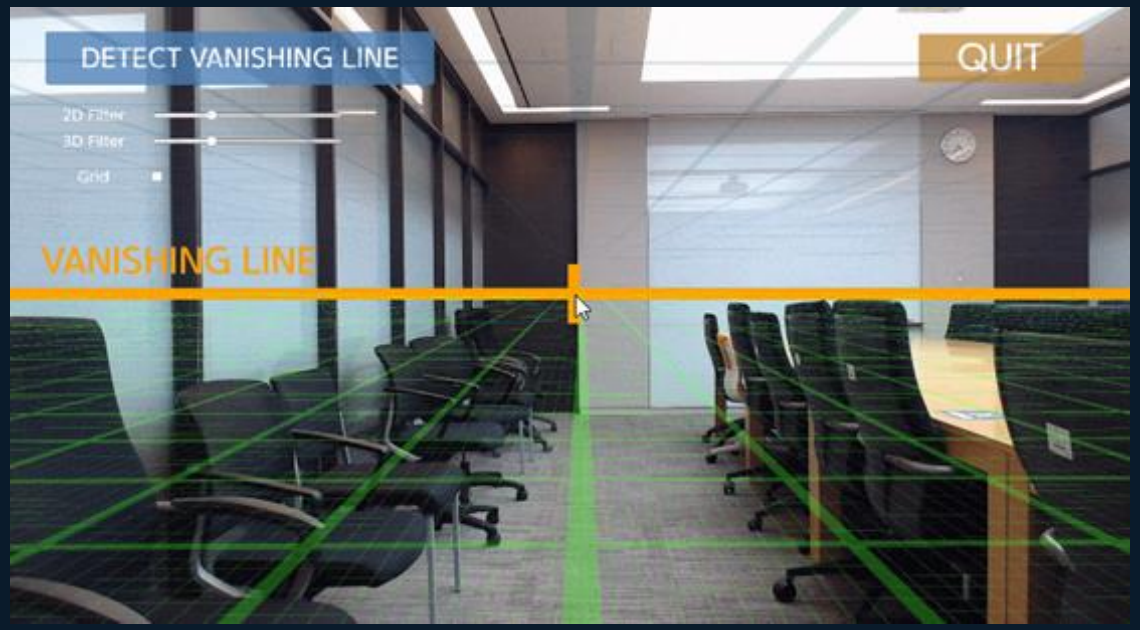
$$q_{initialChildJoint-initialTargetJoint}^{-1} \cdot q_{initialChildJoint-initialTargetJoint} \cdot q_{diff} = q_{initialChildJoint-initialTargetJoint}^{-1} \cdot q_{initialRotation}$$

$$q_{diff} = q_{initialChildJoint-initialTargetJoint}^{-1} \cdot q_{initialRotation}$$

$$q_{updatedRotation} = q_{updatedChildJoint-updatedTargetJoint} \cdot q_{diff}$$

- Presenting 3d keypoints correctly with a deformable character
- Derived quaternion expressions to rotate the character's bones
- Handle different rotations joint by joint
- Can accommodate different characters

Internship project Demo



Determining a ground using a the vanishing point

Human pose estimation

Thank you

Park Changhwi

smsychjy96@gmail.com

https://github.com/ObjectOrientedLife