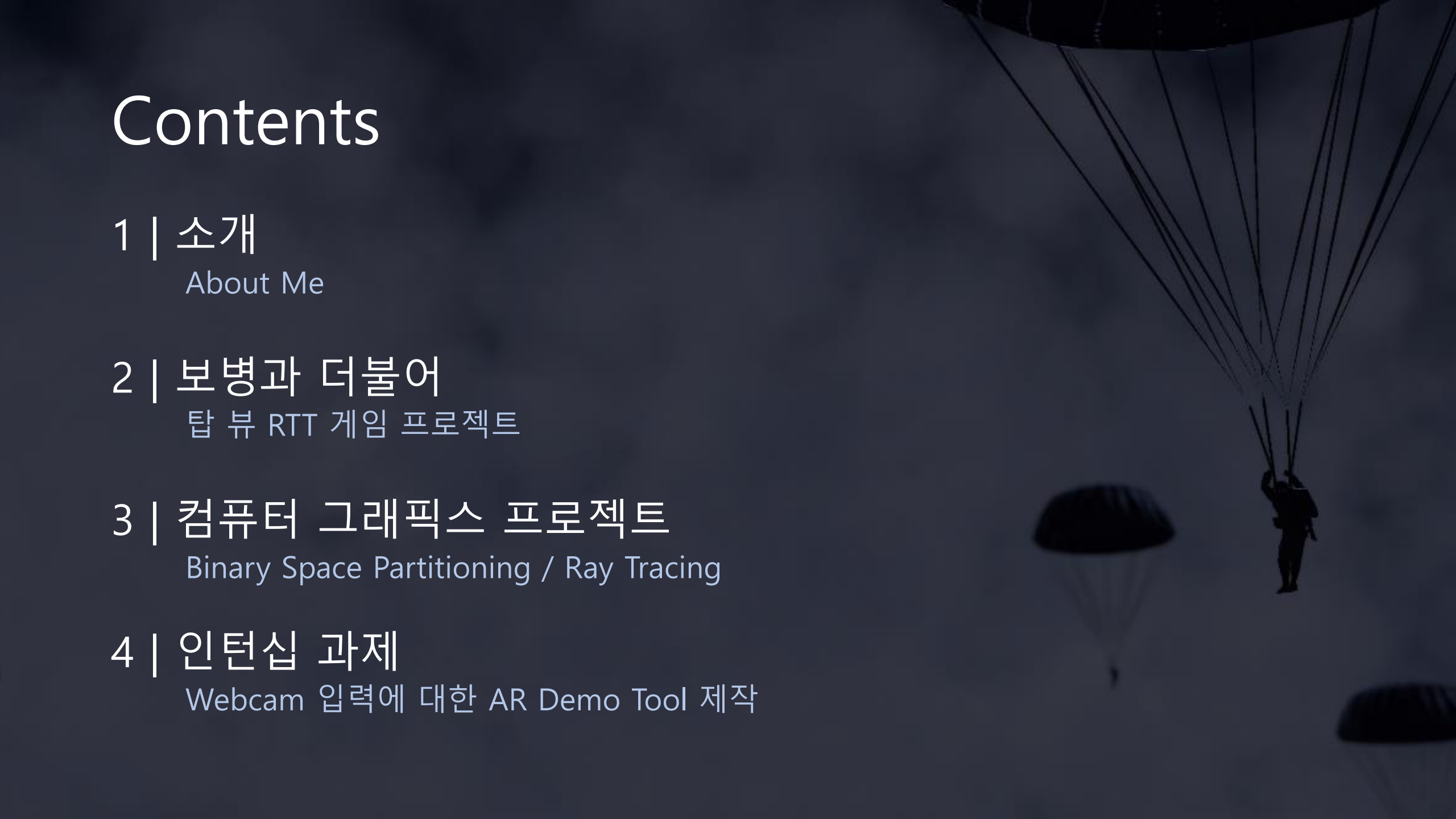
PORTFOLIO

박창휘 Park Changhwi

smsychjy96@gmail.com

https://github.com/ObjectOrientedLife

Contents

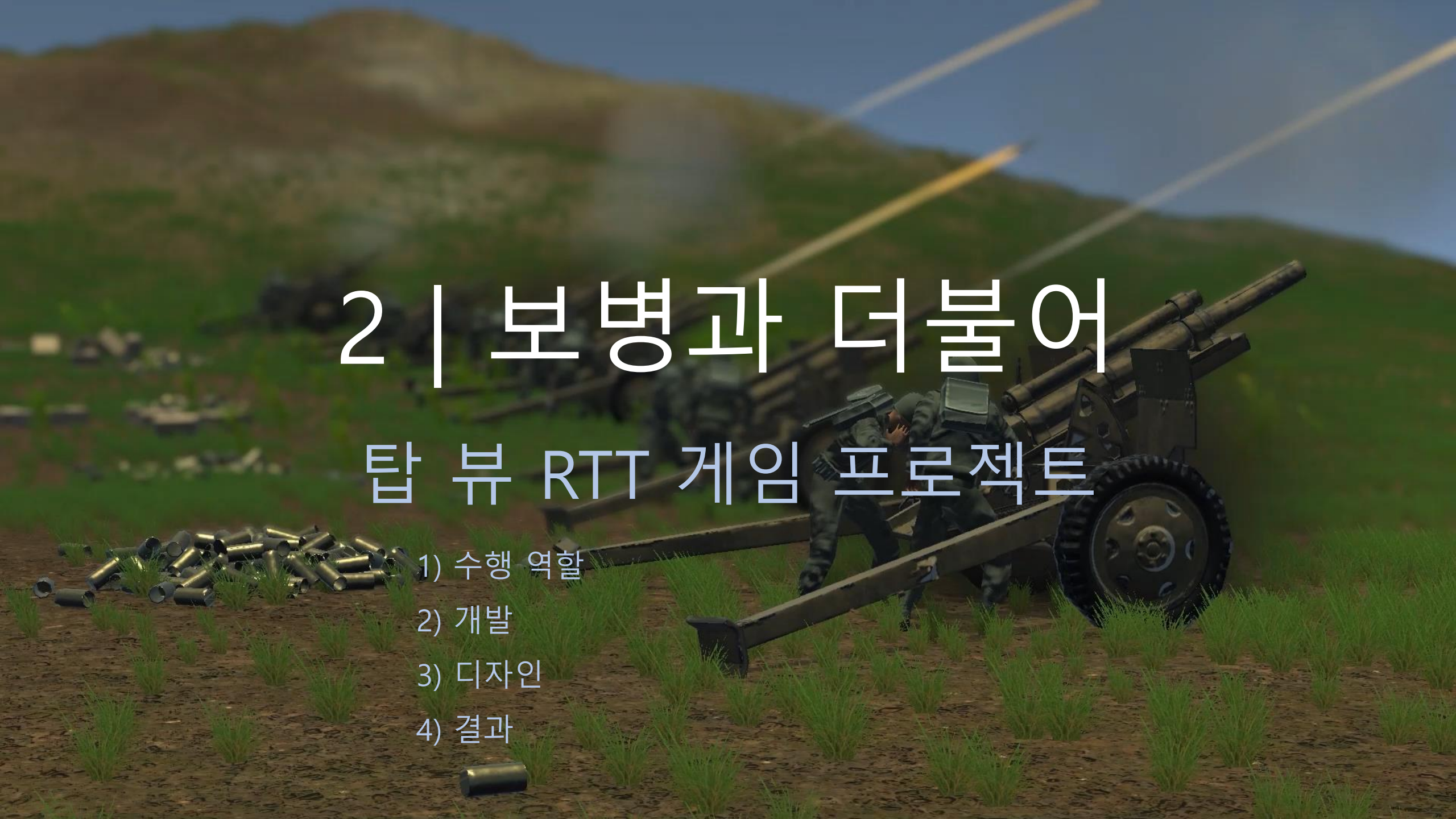


1 | 소개

About Me

소개 About Me

- 박창휘(Park Changhwi)
- smsychjy96@gmail.com
- https://github.com/ObjectOrientedLife
- 서울대학교 지리학과/컴퓨터공학부 복수전공
- 관심 분야 - 게임 전반 / 컴퓨터 그래픽스(실시간 렌더링, 셰이더, 레이 트레이싱, 휴머노이드 애니메이션) / 컴퓨터 비전 / Human pose estimation
- 언어 - C#, C++, Python, Shader language(Cg)
- 스킬 - Unity, OpenGL, OpenCV / Blender, ZBrush, Substance Painter



2 | 보병과 더불어

탑 뷰 RTT 게임 프로젝트

1) 수행 역할
2) 개발
3) 디자인
4) 결과

1) 수행 역할 1인 개발 時

게임 개발 전반의 역할 모두 수행

기획

- 게임 진행 방식 스케치
- 게임 개발 계획 수립
- 배경 조사
- 컨셉 아트

아트

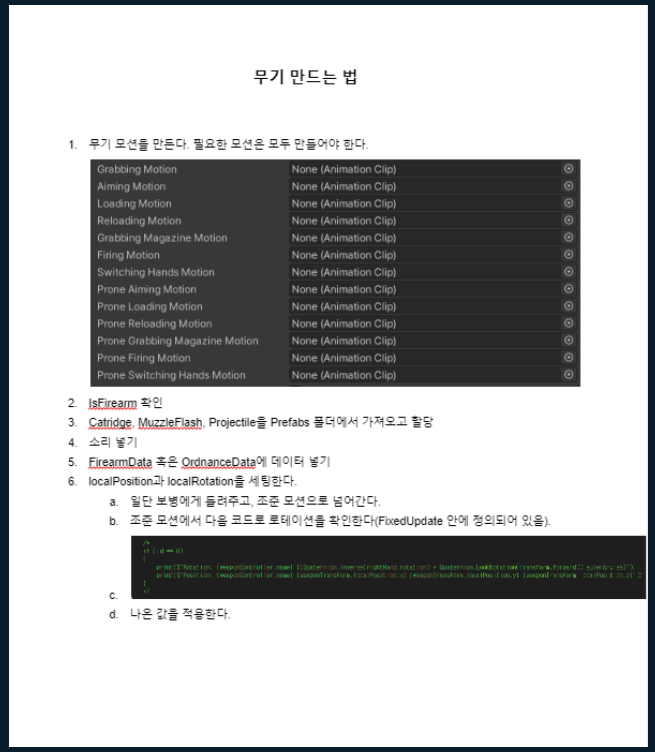
- Modeling
- Sculpting / Texturing
- Rigging / Skinning / Animating

개발

- Unity
- 클라이언트(C#)
- 서버(Unity Mirror)
- 셰이더(Cg)

1) 수행 역할 팀 빌딩 이후

- 1인 개발 -> 3인(프로그래머, 아트2) 팀으로 확충
- 커뮤니케이션 - 미팅 주관, 일정 조율, 기획 의견 수렴, 상호 평가 주도
- 모델링 분담을 위해 3D 모델링 파이프라인 정립
- 학습 영상 및 문서 제작 / 디자인 스펙 작성
- 개발(Unity, C#)과 캐릭터 디자인, 일부 모델링은 그대로 본인이 수행



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 넣어간다.
   b. 조준 모션에서 다음 코드로 로테이션을 확인한다(FixedUpdate 안에 정의되어 있음).

```
/*
if (_d == 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) 개발 클라이언트 전반

문제점

- 다양한 무기(반자동, 자동, 싱글액션)와 장비(항공기, 전차, 야포)가 등장함에 따라 코드가 비대해짐
- 코드의 양이 늘어남에 따라 각 모듈 사이의 상호작용과 작동 방식을 이해하기 어려워짐

- 모바일 환경에서 실행되는 게임인 만큼 최적화의 중요성이 높음
- 가독성 - 추상화 트레이드오프: 추상화 수준을 낮추면 가독성이 저하됨
- 메모리 - 시간 트레이드오프: 오브젝트 풀링은 시간적으로 이득이 있지만 메모리 사용량이 필연적으로 증가함

- 팀 인원이 많지 않은 상황에서 디자인 리소스의 제작 여력에 한계 존재
- 리소스를 매번 새로 만들면 디자인에 일관성을 부여하기 어려움

솔루션

- 재사용성을 높이기 위해 객체지향 프로그래밍 원칙(리스코프 치환 원칙, ...)에 기반한 상속 구조를 설계함
- 디자인 패턴(싱글톤, 추상 팩토리, 전략)을 적극 활용하여 파악 및 수정이 쉽도록 구현과 인터페이스를 분리함

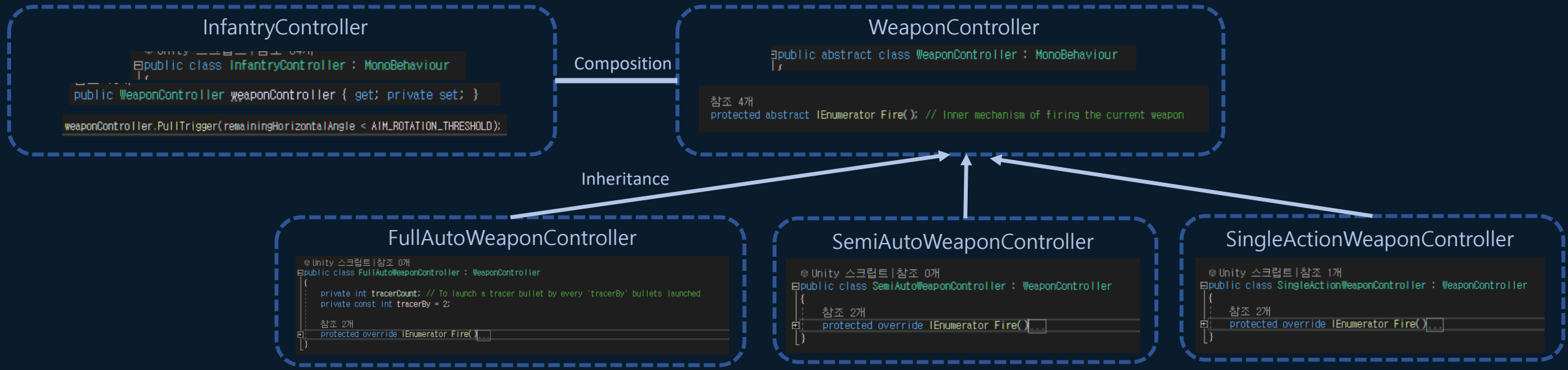
- 게임에서 수행되는 작업들을 4개(게임 최초 실행시 - 플레이 도중 가끔 - 플레이 도중 수시로 - 매 프레임마다) 로 분류하여, 각 분류마다 위계적으로 가독성 - 추상화 / 메모리 - 시간의 트레이드 오프를 적용하고, 그에 알맞은 수준으로 최적화함

- 리소스를 최대한 재사용할 수 있는 코드 작성
- 캐릭터의 경우, 신체 / 복장 / 가젯 세 분류로 나누어 각각 교체할 수 있는 컴포넌트를 개발하여 커스터마이징을 용이하게 함

2) 개발 전략 패턴 - 무기

- 보병은 저마다 하나씩 무기를 들고 있음
- 무기들은 작동 방식과 무관하게 탄환 발사, 총성, 탄피 배출, 장전 등의 공통된 동작을 지니고 있음
- 따라서 모든 무기를 아우르는 WeaponController 추상 클래스를 작성

- 무기를 작동 메커니즘에 따라 자동/반자동/싱글액션으로 분류하고 WeaponController에게 상속받음
- 보병은 무기의 실제 작동 매커니즘을 신경 쓰지 않고 weaponController.PullTrigger() 함수를 호출하여 발사 여부를 결정함
- 구현과 인터페이스의 분리



2) 개발 캐릭터 커스터마이징

- 게임에 다양한 캐릭터를 등장시키면 복장과 신체, 그리고 부착물을 조합하는 경우의 수가 지나치게 증가
- 이를 방지하기 위해 Blender에서 복장, 신체, 부착물을 분리하여 제작한 후, 유니티로 불러와 스크립트를 통해 조합
- Deformable한 오브젝트를 교체한 후, SkinnedMeshRender에서 추가적으로 Bone을 교체하는 작업을 수행
- 부착물은 Deformable하지 않으므로, 태그를 통해 따로 관리

```
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
}
```

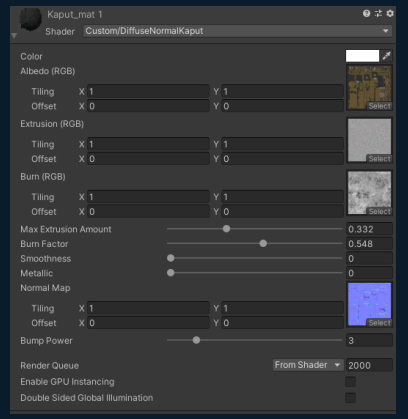
2) 개발 애니메이션 상태 변화 콜백

- Unity는 애니메이션 상태 변화(시작, 종료, 변화)에 따른 콜백을 StateMachineBehaviour으로 지원
- 그러나 StateMachineBehaviour은 기본적으로 런타임에 바꿀 수 없는 구조
- 이를 우회하여 런타임에 애니메이션 상태 변화 콜백을 구독하는 스크립트를 작성
- 콜백 구조를 채용함으로써 애니메이션의 상태 변화를 지속적으로 추적하는 것보다 성능적으로 좋을 뿐만 아니라, 코드가 간결해지고 사용 방법이 직관적으로 변화하는 효과가 발생
- 무기 재장전 모션에 사용
  - 탄창을 잡는 모션이 끝날 때 탄창을 손의 자식 오브젝트로
  - 재장전 모션이 끝나면 탄창을 총의 자식 오브젝트로
  - 재장전 모션이 끝나면 탄약이 보충됨

```
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) 개발 셰이더

- Unity의 기본 셰이딩 언어인 Cg 사용
- 별도의 모델과 텍스처를 제작하지 않고도 파괴된 차량을 묘사할 수 있는 Surface shader 작성
  - Vertex modifier에서는 Noise texture에 따라 Vertex를 Extrude하여 형상을 변형
  - Surface function에서는 Burn texture에 따라 표면을 불탄 것처럼 색을 덧입힘
- Noise texture를 비롯해 인터페이스에 여러 파라미터를 노출시킴으로써 여러 가지 효과를 낼 수 있도록 의도함




```
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);
}
```

일반 셰이더 파괴 효과 셰이더

3) 디자인 일반 모델 / 캐릭터 제작 프로세스

일반 모델

모델 설계
- 구상
- 단순화
- 오브젝트 분리
- 정확한 모델 설계

Blender
- 모델링
- UV unwrapping
- 모델 조립
- Shape animation
- 폴리곤 개수 최적화

ZBrush
- Sculpting
- Subdivision
- ID 맵 Painting
- 섬세한 요철 묘사

Substance Painter
- Texturing
- Diffuse map 및 Normal map export
- 사실감 부여

캐릭터

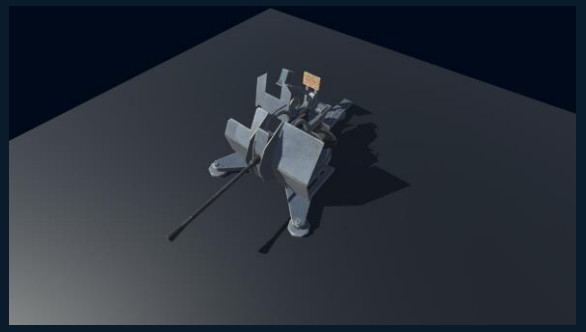
모델 설계
- 신체 / 복장 별도 제작
- 휴대품(모자, 수통 등)도 별도 제작
- 재사용성 높임

모델링(Blender, ZBrush, Substance Painter)
- 캐릭터 모델 제작
- Seam을 숨기는 모델링
- 신체 / 복장 디테일 표현

휴머노이드 제작 (Blender)
- Rigging으로 뼈대 제작
- Skinning으로 캐릭터 메시 연동
- 동작을 정확하게 반영하는 캐릭터 제작

Animating (Blender)
- 캐릭터에 역동성을 부여할 수 있는 다양한 애니메이션
- 유니티에서 오브젝트와 상호작용할 수 있도록 제작

3) 디자인 결과물



3) 결과 트레일러

- https://www.youtube.com/watch?v=b9b-6MzOAi0 (모든 제작 과정을 전담)
- https://www.youtube.com/watch?v=zgcS1foEgOA (일부 무기 및 영상 편집을 제외하고 모두 전담)

3 | 컴퓨터 그래픽스 프로젝트

Binary Space Partitioning / Ray Tracing

컴퓨터 그래픽스 프로젝트 Binary Space Partitioning

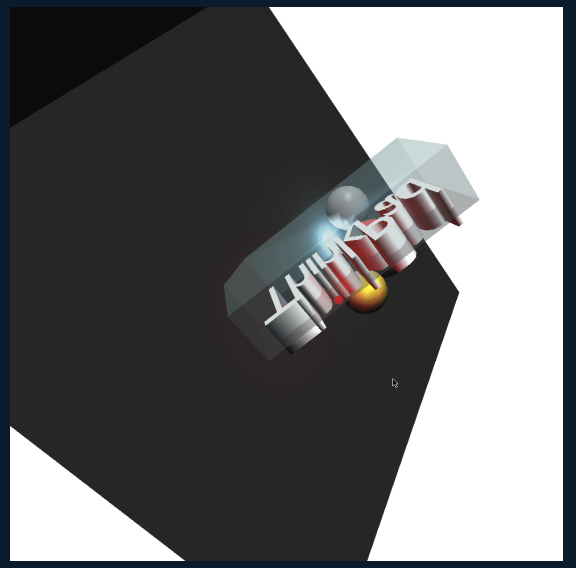
- C++ / OpenGL을 사용하여 구축한 Scene
- 기존 OpenGL의 Depth test로는 투명한 오브젝트 너머로 다른 오브젝트가 비치도록 표현할 수 없음

↓

- Binary Space Partitioning(BSP)을 구현
- 프로그램 시작 시 Tree를 한 번 빌드하고, 매 프레임마다 Traverse
- Traverse하면서 자연스레 정확한 Depth sorting을 수행하게 되며, 어느 시점에서 보더라도 투명한 오브젝트와 그 너머의 오브젝트를 올바르게 표현함

https://github.com/ObjectOrientedLife/BinarySpacePartitioning 에서 소스 코드 제공

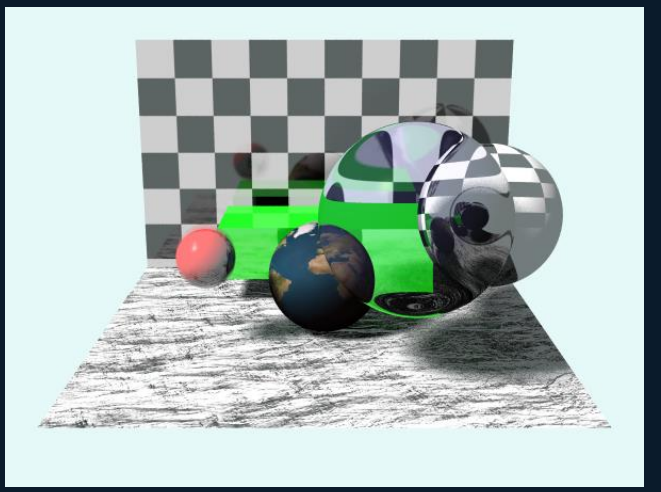
BSP가 구현되지 않은 Scene에서는 투명한 오브젝트 뒤로 다른 오브젝트가 보이지 않음

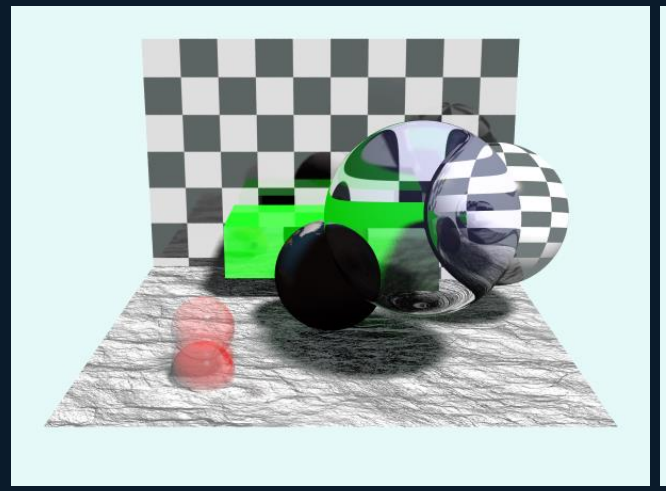


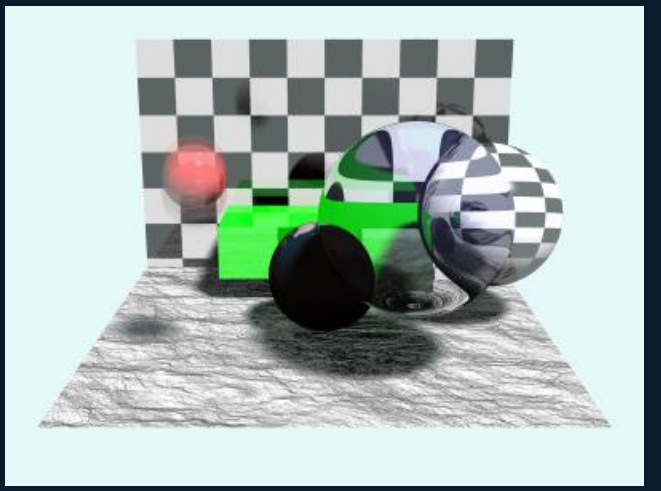
BSP가 구현된 Scene에서는 투명한 오브젝트 뒤로 다른 오브젝트가 정상적으로 렌더링됨

컴퓨터 그래픽스 프로젝트 Ray Tracing

- C++로 구현하였으며, OpenGL과 같은 그래픽 라이브러리는 사용하지 않음
- Phong lighting model을 기반으로 작동하는 Whitted Ray tracing - Recursive refraction, recursive refraction
- Subpixel sampling을 통한 Stochastic ray tracing 구현 - 부드러운 그림자, 질감, 모션 블러
- 텍스처맵 및 노말맵 지원
- .obj파일을 직접 Scene에 배치할 수 있음
- OpenMP에 기반한 Multithreading을 지원하여 멀티 프로세서 환경에서 더욱 빠른 작동(8코어 CPU에서 약 3배 빠름)
- https://github.com/ObjectOrientedLife/RayTracing에서 소스 코드 제공







컴퓨터 그래픽스 프로젝트 Ray Tracing 구현

1. Scene, Light, Shape(Face와 Sphere가 상속)를 각각 클래스로 만들어 추상화 시키고, 별도의 파일에 분리하여 모듈화

```
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
```

Composition

Scene.h - 전체 Scene의 정보를 담고 있는 Scene 클래스와, 그 Scene을 Trace하는 데에 사용될 Ray 및 Hit 구조체를 정의

Shape.h - Shape들이 공통적으로 가지는 정보를 정의

```
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 - Shape를 상속받아 각 Face 혹은 Sphere 개체를 표현함

```
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 - 개별 Light 개체를 나타냄

컴퓨터 그래픽스 프로젝트 Ray Tracing 구현

main.cpp

2. .obj와 텍스처 파일들을 로드

```
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. 각 모듈들을 사용하여 하나의 Scene 구성

```
// ==================== 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. Ray tracing 수행

```
// ==================== 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. 결과 Export

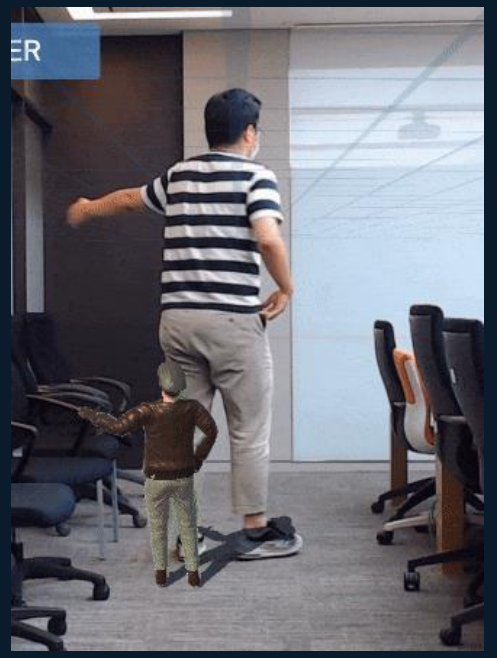
```
vector<vector<vec3>> samples;
for (int i = 0; i < SAMPLE_COUNT; ++i)
{
    samples.push_back(sampleResults[i]);
}
vector<vec3> averaged = getAverageImage(samples); // Take the average

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

4 | 인턴십 과제

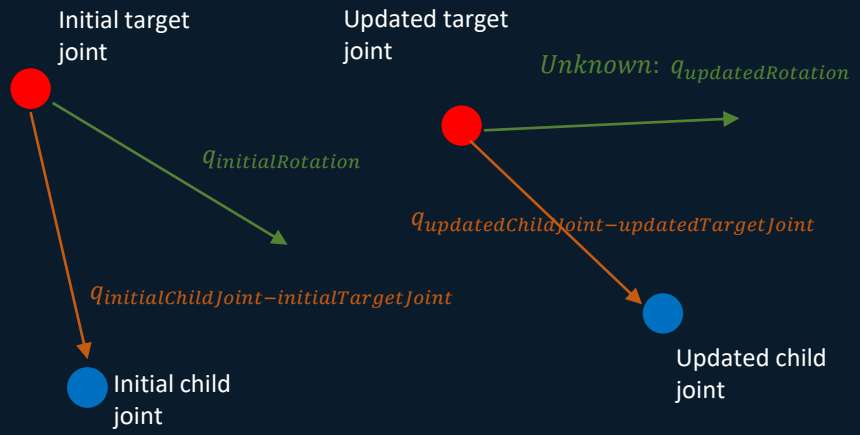
Webcam 입력에 대한 AR Demo Tool 제작

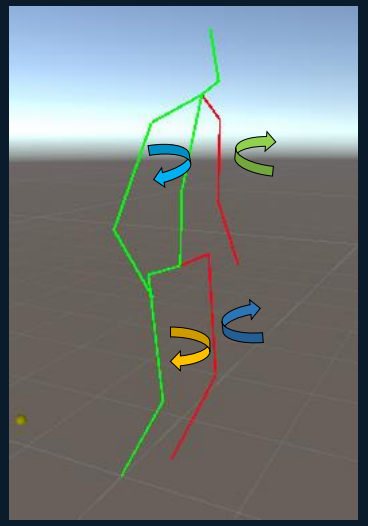
인턴십 과제 Webcam 입력에 대한 AR Demo Tool 제작

- NCSoft / Vision AI Lab / Human Pose Estimation팀
- 2021.07 - 2021.08 두 달 간 과제 수행
- 웹캠을 통해 들어오는 입력을 네트워크에 입력하고, 출력된 Inference 결과를 휴머노이드에 표현
- Inference 결과에 필터를 적용하여 부드러운 움직임 연출
- 다양한 Bone structure, Bone orientation을 가진 제네릭한 휴머노이드 모델에 대응할 수 있도록 구현
- 소실점을 지정 혹은 탐지하는 방식으로 웹캠에서의 증강현실 구현
- 증강현실을 뒷받침하는 그림자 Shader 작성



인턴십 과제 캐릭터 움직임 구현





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

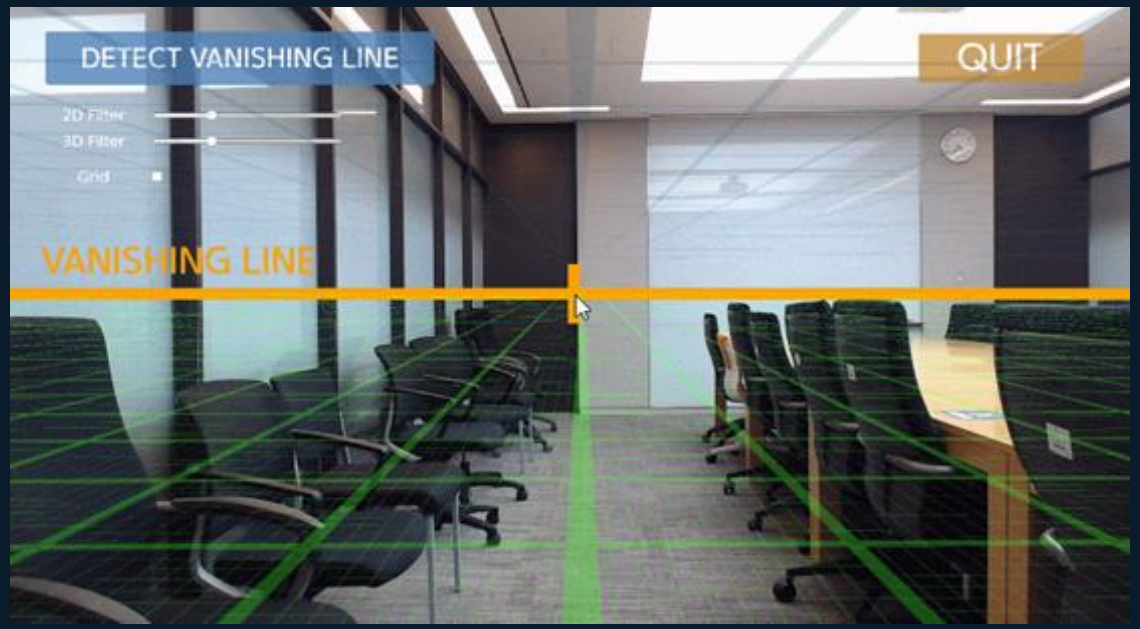
$$q_{initialChildJoint-initialTargetJoint}^{-1} \cdot q_{initialChildJoint-initialTargetJoint} \cdot q_{diff} = q_{initialChildJoint-initialTargetJoint}^{-1} \cdot q_{initialRotation}$$

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

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

- 3D 키포인트의 위치 관계를 Deformable한 캐릭터로 올바르게 나타냄
- 캐릭터의 T-pose를 기준으로 각 관절이 정상적으로 회전하도록 Quaternion 수식 유도 및 구현
- 관절마다 다른 동작을 고려하여 처리(ex - 팔꿈치와 무릎의 회전)
- 여러 가지 캐릭터를 사용할 수 있도록 대응하는 제너릭한 코드 작성

인턴십 과제 시연 결과



소실점을 활용한 AR plane 지정

어플리케이션 작동 결과

감사합니다

박창휘 Park Changhwi

smsychjy96@gmail.com

https://github.com/ObjectOrientedLife