Unity 에서의 GPGPU 기반 광선 추적 렌더러

2019-12-19

SEETHELIGHTS
팀장 김한상 2013154012
김수혁 2012154010
정지윤 2014150031

Index

- 지적사항
- 졸업 연구 개요
- 관련 연구 및 사례
- 시스템 수행 시나리오
- 시스템 구성도
- 모듈 상세 설계
- 개발 환경 및 개발 방법
- 업무 분담
- 졸업 연구 일정
- 필요 기술 및 참고 문헌

지적 사항

C: 입력데이터의 scope를 잘 정의 할 것 구체화 필요

A: 광원, 카메라, 변환, 메터리얼, 텍스쳐 Unity에서 제공하는 데이터로 나눔, 자세한 사항은 24p~

C: 작더라도 확실한 성과가 나올 수 있도록 할 것

A: 각 모듈의 설계를 통한 확실한 목표 설정, 자세한 사항은 55p

C: 목표치에 대한 검증 방법 완성도가 나와야함

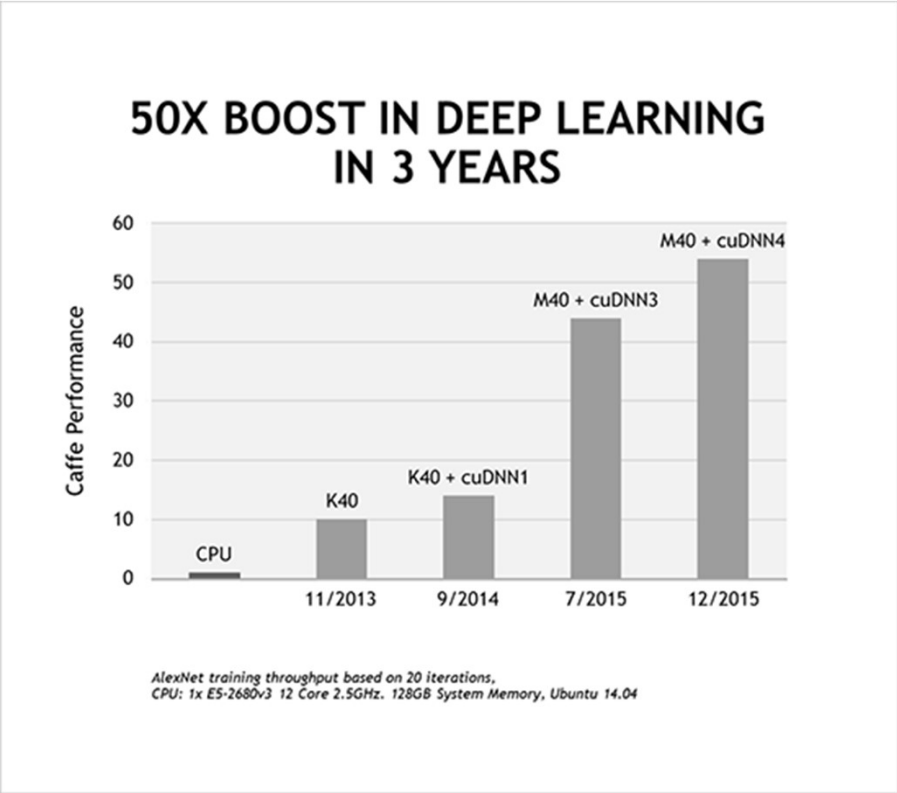
A: 시간을 들여 만든 기준이 되는 이미지와 적정 시간을 들여 만들어진 이미지를 IQA 기법 에러를 계산하여 완성도 측정, 자세한 사항은 48p, 49p

언리얼 엔진 4: 광선 추적 Reflection 데모


EVENT COVERAGE

졸업 연구 개요: 연구 개발 배경

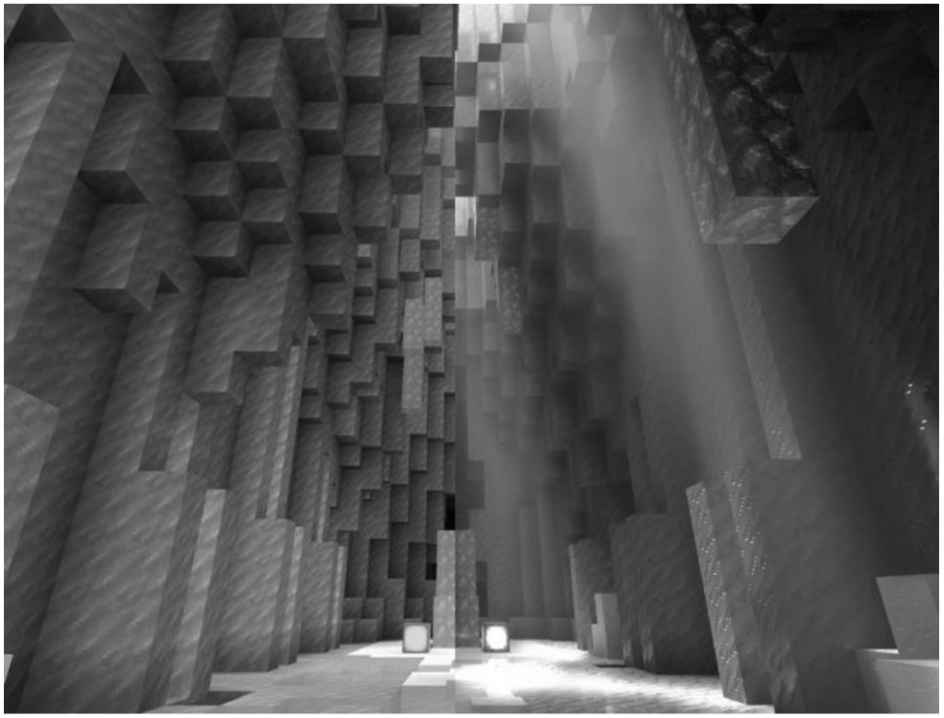
졸업 연구 개요: 연구 개발 배경



AlexNet Performance Profile(CPU, GPGPU – K40, K40+cuDNN1, ..)

- GPU의 범용 계산: GPGPU
  - 독립된 수많은 작업을 수행시 CPU 보다 ***월등한 성능***
    - many-core > multi-core
  - 대표적인 활용: ***AlexNet*** (2012)
  - 다양한 활용: 통계 물리, 신호 처리, 컴퓨터 그래픽, 등..
  - API: CUDA, OpenCL

졸업 연구 개요: 연구 개발 배경



Only Rasterization Minecraft Raytracing(Mirror Reflection) + Rasterization(Multiple Scattering)

- 광선 추적: Ray-Tracing
  - ***<u>빛</u>*** 의 움직임을 ***<u>시뮬레이션</u>*** 하는 컴퓨터 그래픽 기법
  - 이전: 수많은 계산 시간에 외면
  - 현재: GPU의 발달에 편승
    - 실시간 렌더링에서도 부분적으로 사용 (Nvidia RTCore)

졸업 연구 개요: 연구 개발 배경

Youtube: Disney's Practical Guide to Path Tracing

졸업 연구 개요: 연구 개발 배경

- 실시간 렌더링 엔진: Unity, UE4
  - 응답성이 중요한 ***게임 개발*** 용도로 만들어짐
  - VR의 영향으로 ***영상 제작 툴*** 로도 쓰이기 시작
    - 영상제작 S/W 시장의 진입
    - 높은 응답성 -> 작업 능률 상승

졸업 연구 개요: 연구 개발 배경

해외투자 2.0
Global Research

이영진

신뢰에 가치로 답하다 삼성증권 SAMSUNG

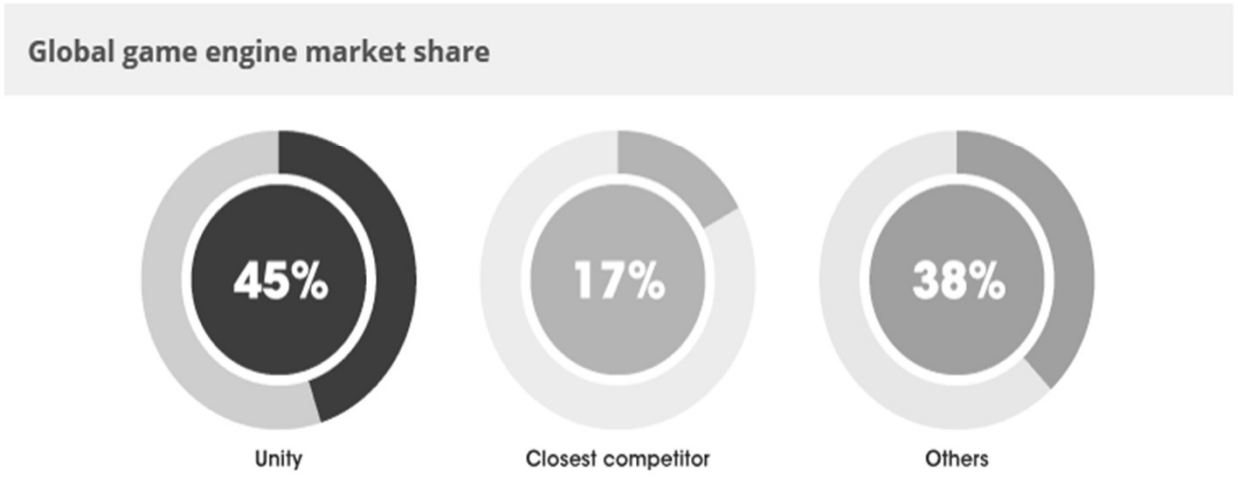
2019. 7. 12

Unity Technologies (Private)

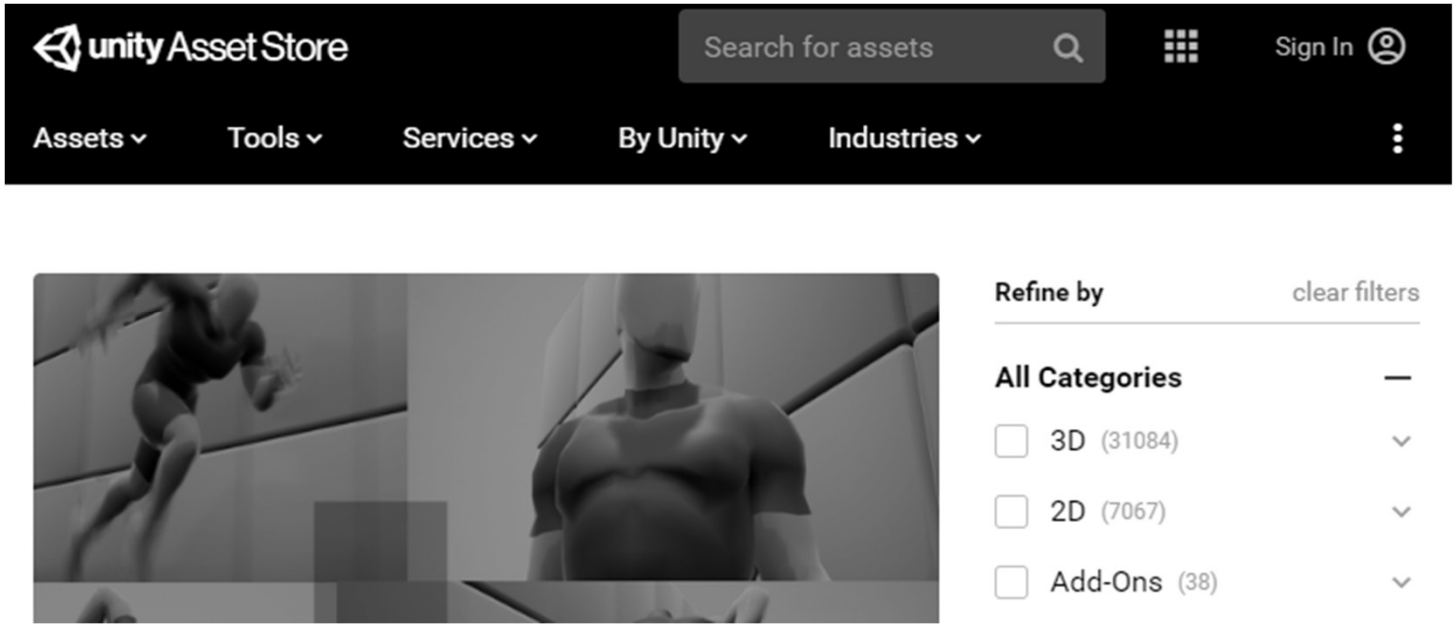
게임 제작에도 족보가 있다

WHAT'S THE STORY

기업소개: 유니티 테크놀로지(이하 유니티)는 유니티 게임 엔진으로 유명한 서드 파티 게임 개발용 소프트웨어 기업으로 유니티 엔진은 세계에서 가장 널리 사용되는 실시



졸업 연구 개요: 연구 개발 배경



Unity Asset Store

Unity 전용 오픈 마켓: 다양한 에셋 판매/구매 가능

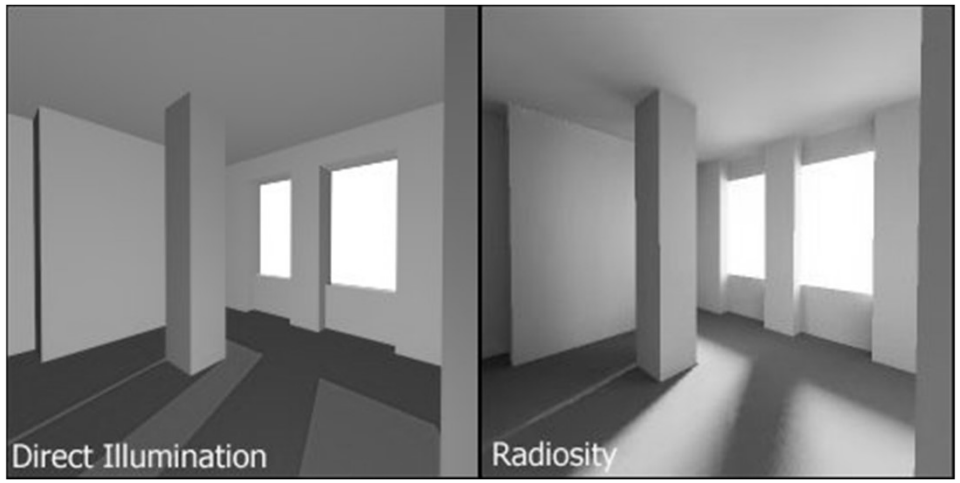
Unity 에서의 즉각적인 사용 가능

졸업 연구 개요: 연구 개발 목표

- 광선의 움직임을 계산하여 사실적인 명암을 가진 이미지 생성
- 연속적인 이미지를 생성하여 동영상 생성
- 하나의 GPU 대비 동일한 성능의 두개의 GPU 사용시 생성 효율 150%~170% 향상
  - 하나의 프레임, HD: 1920 x 1080, RGB color, 500spp

졸업 연구 개요: 연구 개발 효과

영상 제작 툴로 사용시,
빛을 실제와 같이 표현



실제와 비슷한 명암을 모방하기 위한
reference 뷰어로의 활용

관련 연구 및 사례

octanerender

HDRP + DXR

관련 연구 및 사례 : **octane**render

octanerender

- Octane Render
  - GPGPU based path-trace Renderer
  - 대부분의 Game Engine, DCC 들의 기능에 Plugin 으로 지원
  - Path-tracing 이외에도 lightmap baking 등 다양한 Pre-calculation 솔루션 제공
  - ***다른 용도 / 기술적 모티브***
- License
  - Unity: Free(1-GPU)/$20(2-GPU)/$60(N-GPU) per month per seat
  - Other DCC/Engine $699/$899 per year

관련 연구 및 사례 : HDRP in Unity

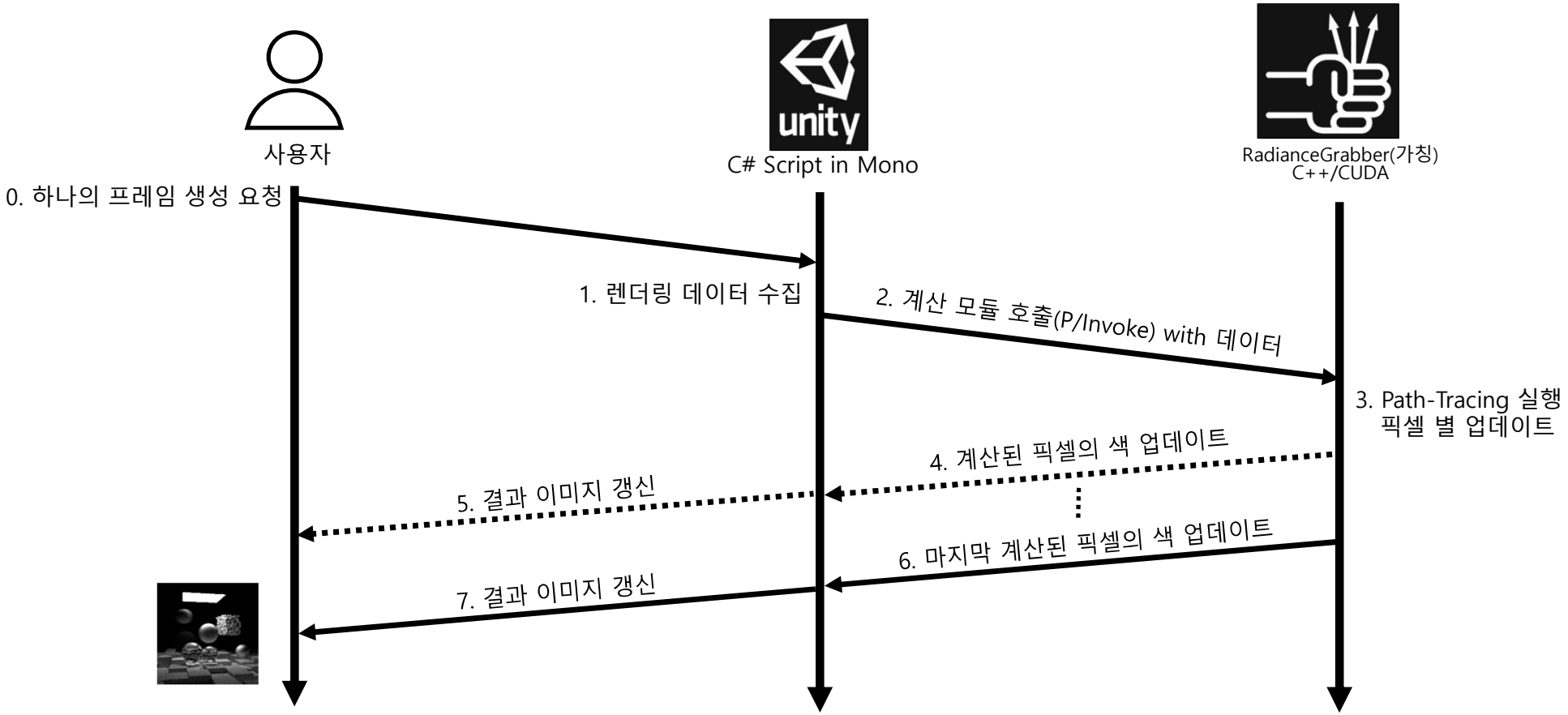
HDRP + DXR

- High Definition RP:
  - Unity 에서 지원하는 오픈소스 프레임워크
  - DXR 을 사용한 Ray Tracing 지원
    - Experimental version
  - ***해당 프레임워크에서만*** Ray Tracing 지원
- DXR: DirectX Raytracing API
  - DirectX 에서 지원하는 기능
    - 2018년 공개(RTX 릴리즈 시기와 동일)
  - ***최신 GPU 에서만*** 기능 지원

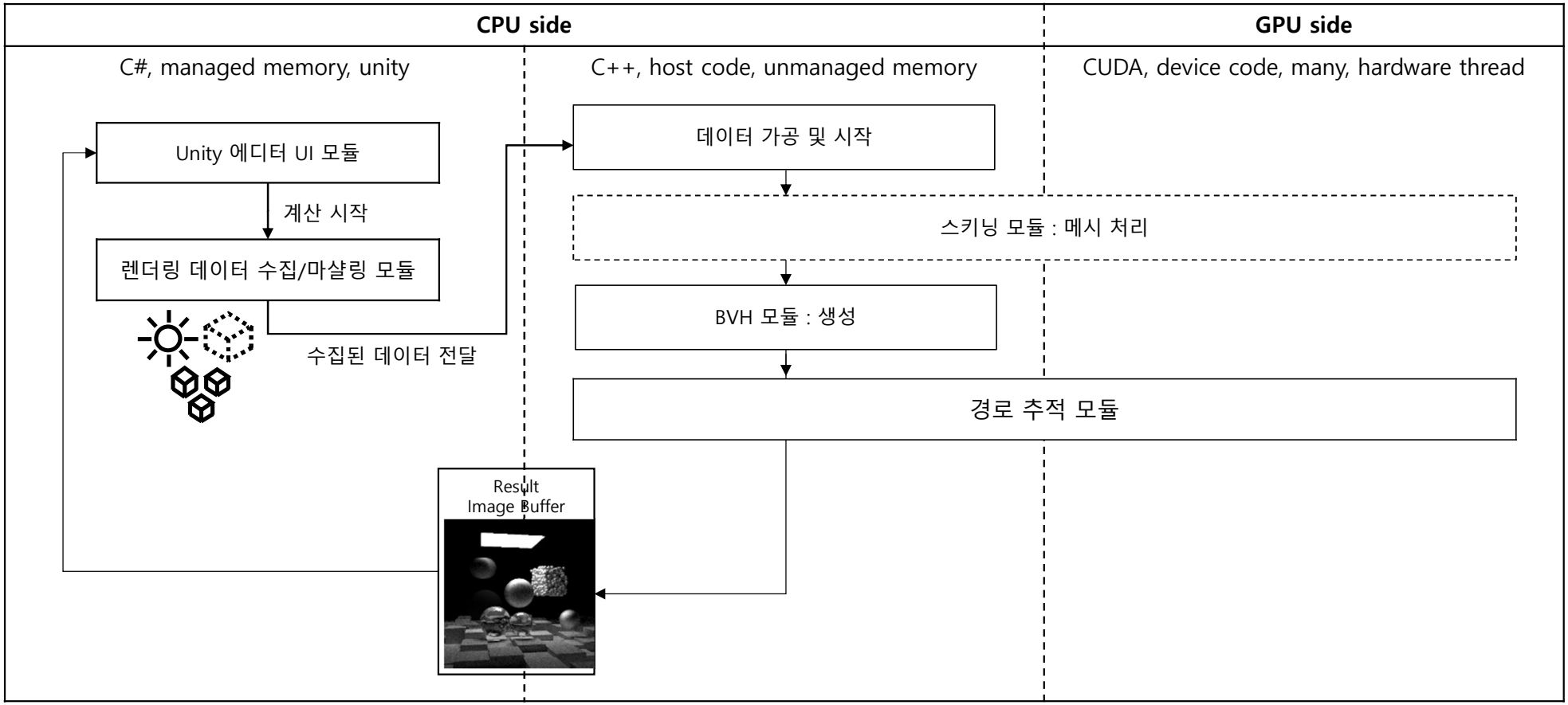
관련 연구 및 사례

	목적	프레임워크 제한	H/W 제한
octanerender	Baking Solution	X	X
HDRP + DXR	Ray-Tracing Render	HDRP 전용	DXR 지원 H/W
RadianceGrabber	Ray-Tracing Render	X	NVidia GPU

시스템 수행 시나리오: 하나의 프레임



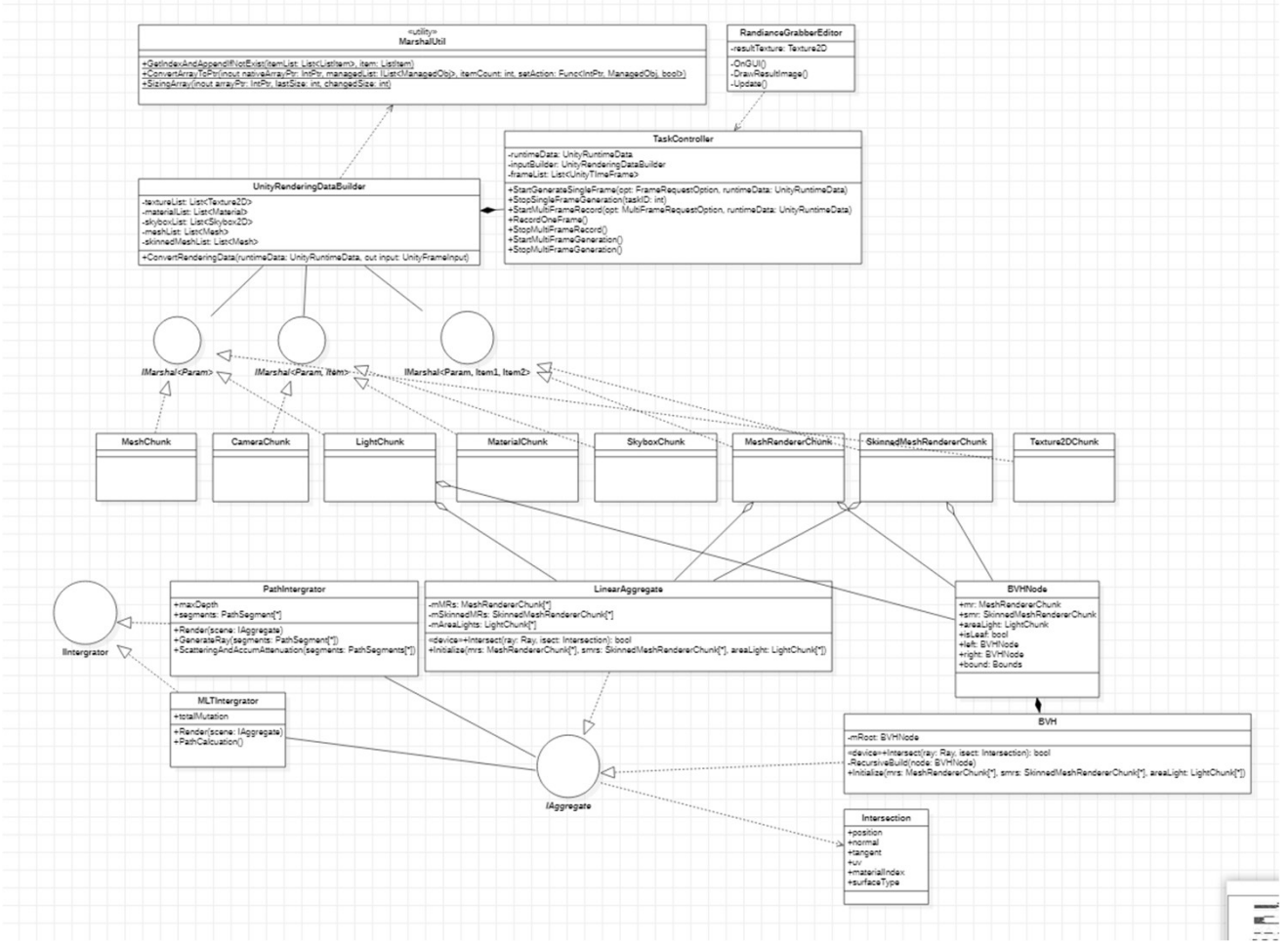
시스템 구성도

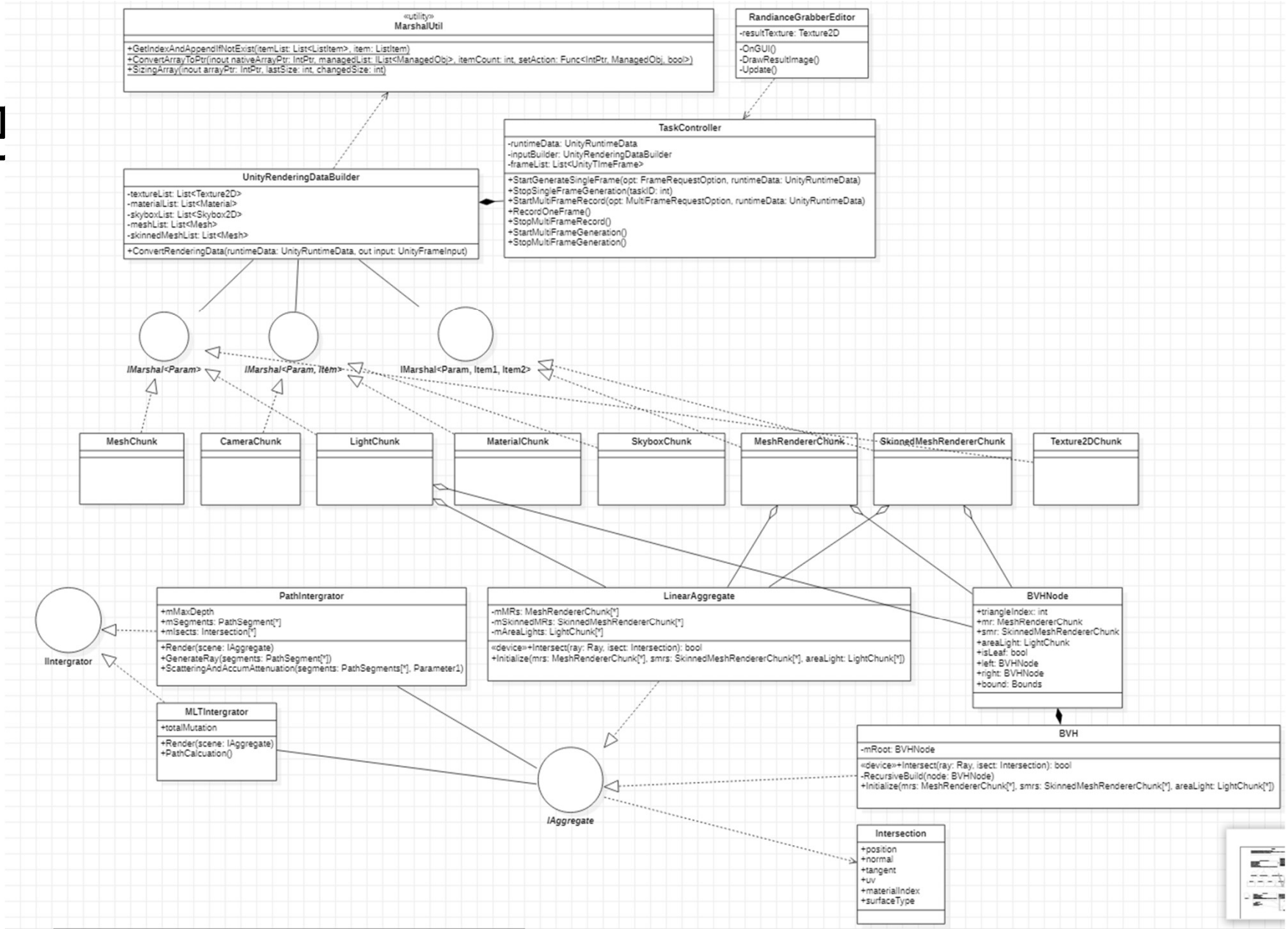

CPU side
GPU side
C#, managed memory, unity
C++, host code, unmanaged memory
CUDA, device code, many, hardware thread
Unity 에디터 UI 모듈
계산 시작
렌더링 데이터 수집/마샬링 모듈
수집된 데이터 전달
데이터 가공 및 시작
스키닝 모듈 : 메시 처리
BVH 모듈 : 생성
경로 추적 모듈
Result
Image Buffer

모듈 상세 설계

- 입력 데이터 마샬링
- Path Tracing, MLT
- BVH
- 목표치 검증

모듈 상세 설계 : 클래스 다이어그램

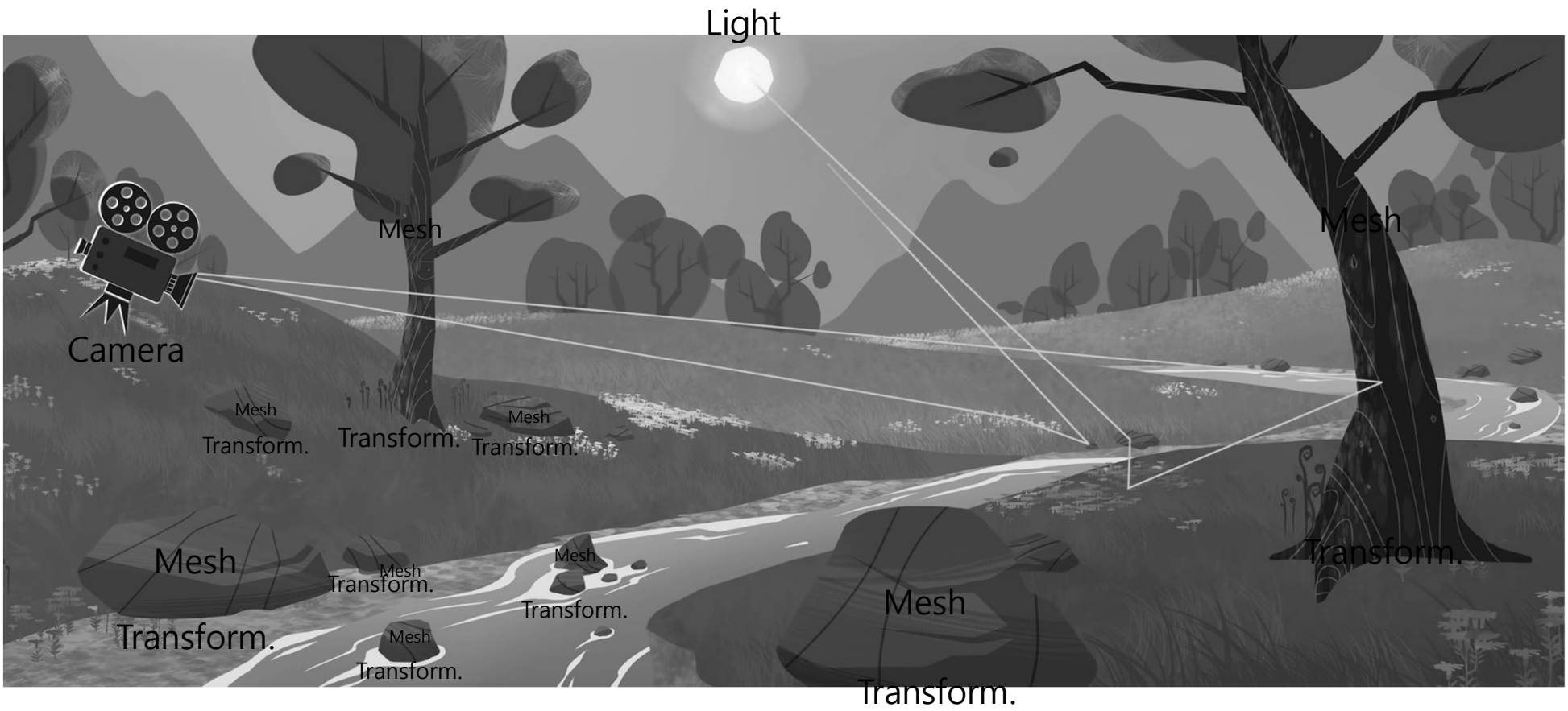



«utility»
MarshalUtil
+GetIndexAndAppendIfNotExist(itemList: List<ListItem>, item: ListItem)
+ConvertArrayToPtr(inout nativeArrayPtr: IntPtr, managedList: IList<ManagedObj>, itemCount: int, setAction: Func<IntPtr, ManagedObj, bool>)
+SizingArray(inout arrayPtr: IntPtr, lastSize: int, changedSize: int)
RandianceGrabberEditor
-resultTexture: Texture2D
-OnGUI()
-DrawResultImage()
-Update()
TaskController
-runtimeData: UnityRuntimeData
-inputBuilder: UnityRenderingDataBuilder
-frameList: List<UnityTImeFrame>
+StartGenerateSingleFrame(opt: FrameRequestOption, runtimeData: UnityRuntimeData)
+StopSingleFrameGeneration(taskID: int)
+StartMultiFrameRecord(opt: MultiFrameRequestOption, runtimeData: UnityRuntimeData)
+RecordOneFrame()
+StopMultiFrameRecord()
+StartMultiFrameGeneration()
+StopMultiFrameGeneration()
UnityRenderingDataBuilder
-textureList: List<Texture2D>
-materialList: List<Material>
-skyboxList: List<Skybox2D>
-meshList: List<Mesh>
-skinnedMeshList: List<Mesh>
+ConvertRenderingData(runtimeData: UnityRuntimeData, out input: UnityFrameInput)
IMarshal<Param>
IMarshal<Param, Item>
IMarshal<Param, Item1, Item2>
MeshChunk
CameraChunk
LightChunk
MaterialChunk
SkyboxChunk
MeshRendererChunk
SkinnedMeshRendererChunk
Texture2DChunk
IIntergrator
PathIntergrator
+mMaxDepth
+mSegments: PathSegment[*]
+mIsects: Intersection[*]
+Render(scene: IAggregate)
+GenerateRay(segments: PathSegment[*])
+ScatteringAndAccumAttenuation(segments: PathSegments[*], Parameter1)
LinearAggregate
-mMRs: MeshRendererChunk[*]
-mSkinnedMRs: SkinnedMeshRendererChunk[*]
-mAreaLights: LightChunk[*]
«device»+Intersect(ray: Ray, isect: Intersection): bool
+Initialize(mrs: MeshRendererChunk[*], smrs: SkinnedMeshRendererChunk[*], areaLight: LightChunk[*])
BVHNode
+triangleIndex: int
+mr: MeshRendererChunk
+smr: SkinnedMeshRendererChunk
+areaLight: LightChunk
+isLeaf: bool
+left: BVHNode
+right: BVHNode
+bound: Bounds
MLTIntergrator
+totalMutation
+Render(scene: IAggregate)
+PathCalcuation()
BVH
-mRoot: BVHNode
«device»+Intersect(ray: Ray, isect: Intersection): bool
-RecursiveBuild(node: BVHNode)
+Initialize(mrs: MeshRendererChunk[*], smrs: SkinnedMeshRendererChunk[*], areaLight: LightChunk[*])
IAggregate
Intersection
+position
+normal
+tangent
+uv
+materialIndex
+surfaceType

모듈 상세 설계 : 입력 데이터 마샬링

- Unity 렌더링 데이터
  - Light
    - 광원, 렌더링의 기준
  - Camera
    - 3D 공간의 기준
  - Transformation
    - 물체들의 위치
  - Mesh
    - 오브젝트의 기하학적 정보(정점, 폴리곤, uv,..)
  - Material
    - 오브젝트의 재질 정보(BSDF)
  - Texture
    - 재질 정보의 1D,2D,3D LUT

모듈 상세 설계 : 입력 데이터 마샬링



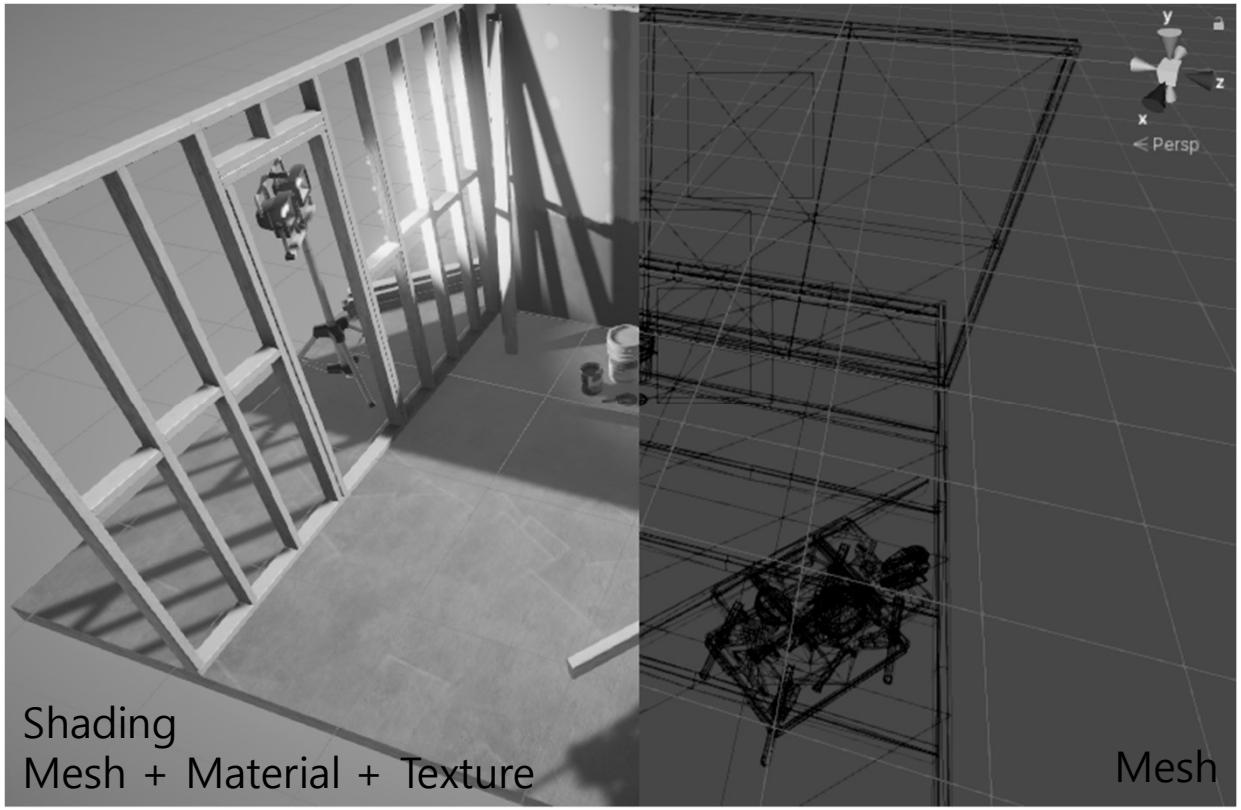
모듈 상세 설계 : 입력 데이터 마샬링

- Material

Mesh와 함께 사용되는 표면 재질의 표현 방식의 인스턴스

BSDF: 표면의 특징

Texture(LUT)와 같은 인스턴스 정보



- Mesh

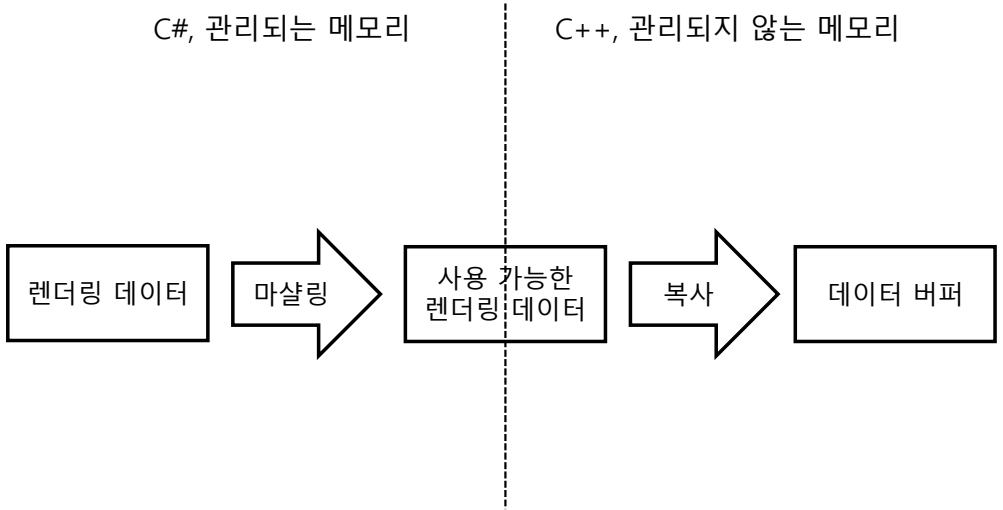
물체의 기하학적 정보의 총체적 집합

정점 별 데이터 : 위치, normal, tangent, uv, ..

Plane: 3개 이상의 정점들의 집합

모듈 상세 설계 : 입력 데이터 마샬링

- Unity 렌더링 데이터?
  - C++ 에서 사용하기 위해선 변환 과정이 필요
- 마샬링
  - 서로 다른 메모리 구조를 가진 언어들이 한쪽으로 데이터를 전달하기 위해 변환하는 것



모듈 상세 설계 : 입력 데이터 마샬링

메소드	**ConvertRenderingData()**
형식	public bool ConvertRenderingData (UnityRuntimeData runtimeData, out UnityFrameInput inputData)
반환 값	성공 시 True, 실패 시 False
설명	Unity 렌더링 데이터를 C++로 넘길 수 있도록 변환
예시	inputBuilder.ConvertRenderingData(ref req.inputData, runtimeData);

ConvertRenderingData() 데이터 구조		
데이터	자료형	설명
runtimeData	UnityRuntimeData	Unity 내에서 수집한 렌더링 데이터의 원본
inputData	UnityFrameInput(out)	마샬링한 데이터를 넣어서 전달할 변수
함수		
ConvertArrayToPtr	C# Array를 Unmanaged memory 의 형태로 IntPtr 로 반환해주는 함수	
GetIndexAndAppendIfNotExist	List에 데이터가 있으면 그 인덱스를 반환하고, 없으면 추가하며 마지막 인덱스를 반환하는 함수	

모듈 상세 설계 : 입력 데이터 마샬링

MarshalFrom			
형식	public bool MarshalFrom(Mesh ms)	설명	메시 데이터를 C++로 넘길 수 있도록 변환
반환 값	성공 시 True, 실패 시 False	예시	meshChunkArray[i].MarshalFrom(meshArray[i]);

데이터구조

데이터	자료형	설명	함수	
ms	Mesh	Unity에서의 메쉬 데이터	Marshal.Copy Memory	unsafe IntPtr과 C# Array간의 데이터 복사를 수행

MarshalFrom			
형식	public bool MarshalFrom(MeshRenderer mr, List<Mesh> meshList, List<Material> materialList)	설명	메시 렌더러 데이터를 C++로 넘길 수 있도록 변환
반환 값	성공 시 True, 실패 시 False	예시	meshRendererChunkArray[i].MarshalFrom(mesRendererhArray[i]);

데이터구조

데이터	자료형	설명	함수	
ms	Mesh	Unity에서의 메쉬 데이터	Marshal.Copy Memory	unsafe IntPtr과 C# Array간의 데이터 복사를 수행
meshList	List<Mesh>	MeshRenderer가 참조하는 메쉬의 리스트, 메소드 안에서는 추가하기만 함		
materialList	List<Material>	MeshRenderer가 참조하는 메터리얼의 리스트, 메소드 안에서는 추가하기만 함		

모듈 상세 설계 : 입력 데이터 마샬링

MarshalFrom			
형식	public bool MarshalFrom(Texture tex)	설명	2차원 텍스쳐 데이터를 C++로 넘길 수 있도록 변환
반환 값	성공 시 True, 실패 시 False	예시	texture2DChunkArray[i].MarshalFrom(texArray[i]);

데이터구조				
데이터	자료형	설명	함수	
tex	Texture	Unity에서의 텍스쳐 데이터	Marshal.Copy Memory	unsafe IntPtr과 C# Array간의 데이터 복사를 수행

MarshalFrom			
형식	public bool MarshalFrom(SkinnedMeshRenderer smr, List<Mesh> meshList, List<Material> materialList)	설명	스킨드 메시 렌더러 데이터를 C++로 넘길 수 있도록 변환
반환 값	성공 시 True, 실패 시 False	예시	skinnedMeshRendererChunkArray[i].MarshalFrom(skinnedMeshRendererArray[i]);

데이터구조				
데이터	자료형	설명	함수	
smr	SkinnedMeshRenderer	Unity에서의 스킨 메시 렌더러 객체	Marshal.Copy Memory	unsafe IntPtr과 C# Array간의 데이터 복사를 수행
meshList	List<Mesh>	MeshRenderer가 참조하는 메쉬의 리스트, 메소드 안에서는 추가하기만 함		
materialList	List<Material>	MeshRenderer가 참조하는 메터리얼의 리스트, 메소드 안에서는 추가하기만 함		

모듈 상세 설계 : 입력 데이터 마샬링

MarshalFrom			
형식	public bool MarshalFrom(Light light)	설명	광원 데이터를 C++로 넘길 수 있도록 변환
반환 값	성공 시 True, 실패 시 False	예시	lightChunkArray[i].MarshalFrom(lightArray[i]);

데이터구조				
데이터	자료형	설명	함수	
light	Light	Unity에서의 광원 데이터	Marshal.Copy Memory	unsafe IntPtr과 C# Array간의 데이터 복사를 수행

MarshalFrom			
형식	public bool MarshalFrom(Camera cam, List<Material> skyboxList)	설명	카메라 데이터를 C++로 넘길 수 있도록 변환
반환 값	성공 시 True, 실패 시 False	예시	cameraChunkArray[i].MarshalFrom(cameraArray[i]);

데이터구조				
데이터	자료형	설명	함수	
ms	Camera	Unity에서의 카메라 객체	Marshal.Copy Memory	unsafe IntPtr과 C# Array간의 데이터 복사를 수행
materialList	List<Material>	Camera가 참조하는 스카이박스 메터리얼의 리스트, 메소드 안에서는 추가하기만 함		

모듈 상세 설계 : 입력 데이터 마샬링

MarshalFrom			
형식	public bool MarshalFrom(Material mat, List<Texture> texList)	설명	메터리얼 데이터를 C++로 넘길 수 있도록 변환
반환 값	성공 시 True, 실패 시 False	예시	materialChunkArray[i].MarshalFrom(materialArray[i]);

데이터구조				
데이터	자료형	설명	함수	
mat	Material	Unity에서의 광원 데이터	Marshal.Copy Memory	unsafe IntPtr과 C# Array간의 데이터 복사를 수행
texList	List<Texture>	Material이 참조하는 텍스쳐의 리스트, 메소드 안에서는 추가하기만 함		

MarshalFrom			
형식	public bool MarshalFrom(Skybox sky, List<texture> texList)	설명	스카이박스 데이터를 C++로 넘길 수 있도록 변환
반환 값	성공 시 True, 실패 시 False	예시	skyboxChunkArray[i].MarshalFrom(sykboxArray[i]);

데이터구조				
데이터	자료형	설명	함수	
sky	Skybox	Unity에서의 카메라 객체	Marshal.Copy Memory	unsafe IntPtr과 C# Array간의 데이터 복사를 수행
texList	List<Texture>	Skybox에서 참조하는 텍스쳐의 리스트, 메소드 안에서는 추가하기만 함		

모듈 상세 설계 : 입력 데이터 마샬링

메소드	**StartSingleFrameGeneration()**
형식	int StartSingleFrameGeneration(FrameRequestOption opt, UnityRuntimeData runtimeData)
반환 값	해당 Task의 ID
설명	하나의 프레임 생성을 위한 DLL안의 함수 호출
예시	int taskID = taskController.StartSingleFrameGeneration(opt, runtimeData);

StartSingleFrameGeneration() 데이터 구조		
데이터	자료형	설명
opt	FrameRequestOption	프레임 생성시 옵션 값의 모음
runtimeData	UnityRuntimeData	Unity의 런타임 렌더링 정보

메소드	**StopSingleFrameGeneration()**
형식	public bool StopSingleFrameGeneration(int taskID)
반환 값	찾아서 멈추었으면 True, 아니면 False
설명	넘겨받은 task ID로 진행되던 이미지 생성을 중단
예시	If (inputBuilder.ConvertRenderingData(currentTaskID)) Debug.LogFormat("이미지 생성({0})이 중지되었습니다.", currentTaskID);

StopSingleFrameGeneration() 데이터 구조		
데이터	자료형	설명
taskID	int	멈출 Task ID, Single Frame 에서 할당된 것만 해당

모듈 상세 설계 : 입력 데이터 마샬링

메소드	**StartMultiFrameRecord()**
형식	int StartMultiFrameRecord(MultiFrameRequestOption opt, UnityRuntimeData runtimeData)
반환 값	해당 Task의 ID
설명	여러 개의 프레임 생성을 위한 데이터 저장 준비
예시	int taskID = taskController.StartMultiFrameRecord(mopt, runtimeData);

StartMultiFrameRecord() 데이터 구조		
데이터	자료형	설명
mopt	MultiFrameRequestOption	여러 개의 프레임 생성시 옵션 값의 모음
runtimeData	UnityRuntimeData	Unity의 런타임 렌더링 정보

메소드	**StopMultiFrameRecord()**
형식	public bool StopMultiFrameRecord (int taskID)
반환 값	찾아서 멈추었으면 True, 아니면 False
설명	넘겨받은 task ID로 진행되던 이미지 생성을 중단
예시	taskController.StopMultiFrameRecord(selectedTaskID);

StopMultiFrameRecord() 데이터 구조		
데이터	자료형	설명
taskID	int	멈출 Task ID, Single Frame에서 할당된 것만 해당

모듈 상세 설계 : 입력 데이터 마샬링

메소드	**RecordOneFrame()**
형식	void **RecordOneFrame** ()
설명	여러 개의 프레임 생성을 위한 현재 상태의 바뀔 수 있는 렌더링 정보를 저장
예시	taskController.RecordOneFrame();

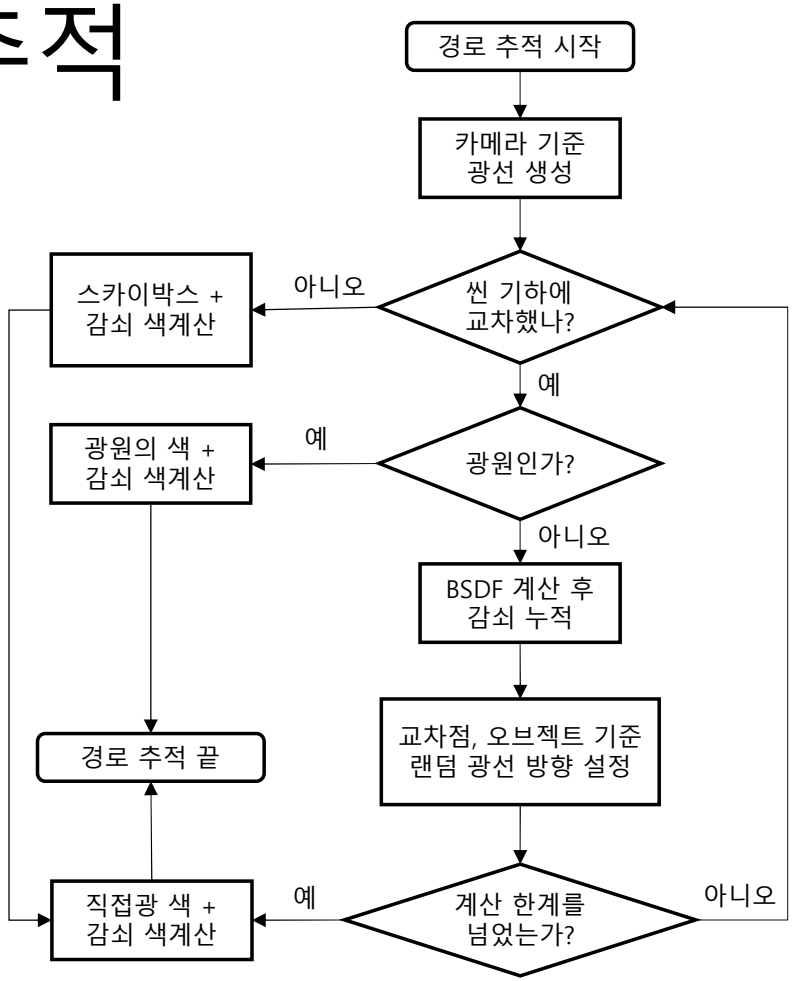
메소드	**StartMultiFrameGeneration()**
형식	int StartMultiFrameGeneration()
반환 값	해당 Task의 ID
설명	여러 개의 프레임 생성을 위한 DLL안의 함수 호출
예시	int taskID = taskController.StartMultiFrameGeneration();

메소드	**StopMultiFrameGeneration()**
형식	public bool **StopMultiFrameGeneration** (int taskID)
반환 값	찾아서 멈추었으면 True, 아니면 False
설명	여러 개의 프레임 생성 취소를 위한 DLL안의 함수 호출
예시	If (taskController.StopMultiFrameGeneration(selectedTaskID);) Debug.LogFormat("이미지 생성({0})이 중지되었습니다.", selectedTaskID);

StopMultiFrameGeneration() 데이터 구조		
데이터	자료형	설명
taskID	int	멈출 Task ID, Single Frame에서 할당된 것만 해당

모듈 상세 설계 : 경로 추적

- 경로 추적 : Path Tracing
  - 광선의 길을 추적하여 색을 계산하는 기법
    - 핵심적인 알고리즘
  - Uniform random sampling
    - 레퍼런스 이미지 생성용
  - 샘플링 횟수 대비 노이즈 多
    - 노이즈를 줄이기 위해선 엄청 많은 샘플링 수가 필요



모듈 상세 설계 : 경로 추적

메소드	**Render()**
형식	void Render(const IAggregate& scene)
설명	주어진 정보를 가지고 Path-Tracing 을 수행, IIntergrator 구현
예시	path.Render(bvh);

Render() 데이터 구조		
데이터	자료형	설명
scene	const IAggregate&	모든 부딪칠 수 있는 물체들을 검사할 수 있는 추상 클래스의 인스턴스(오브젝트, 광원, ..etc)

메소드	**GenerateRay()**
형식	__global__ void GenerateRay(PathSegment* semgents)
설명	카메라를 기준으로 광선 방향과 위치 설정, GPU 커널 실행 함수
예시	GenerateRay<<<optimalGridDim, optimalBlockDim>>>(mSegments);

GenerateRay() 데이터 구조		
데이터	자료형	설명
semgents	PathSegment*	경로 추적을 위한 적은 메모리 공간으로 각 PathSegment 는 픽셀하나 당 계산을 위해 사용, 현 메소드에서는 광선을 정하기 위한 용도

모듈 상세 설계 : 경로 추적

메소드	**ScatteringAndAccumAttenuation()**
형식	__global__ void ScatteringAndAccumAttenuation(PathSegment* semgents, Intersection* isects)
설명	광선이 표면에 부딪친 정보를 통해 광선 방향과 감쇠 계수 설정, GPU 커널 실행 함수
예시	ScatteringAndAccumAttenuation<<<optimalGridDim, optimalBlockDim>>>(mSegments, mIsects);

ScatteringAndAccumAttenuation() 데이터 구조		
데이터	자료형	설명
semgents	PathSegment*	경로 추적을 위한 적은 메모리 공간으로 각 PathSegment 는 픽셀하나 당 계산을 위해 사용
isects	Intersection*	광선이 표면에 부딪친 부분에 대한 정보, 이를 통해 픽셀의 색이 결정됨.

모듈 상세 설계 : 경로 추적

- MLT 계산 모듈
  - Path-Tracing은
- Path Tracing 계산 모듈 vs MLT 계산 모듈

Table 16.1: Percentage of Traced Paths That Carried Zero Radiance. With these two scenes, both path tracing and BDPT have trouble finding light-carrying paths: the vast majority of the generated paths don't carry any radiance at all. Thanks to local exploration, Metropolis is better able to find additional light-carrying paths after one has been found. This is one of the reasons why it is more efficient than those approaches here.

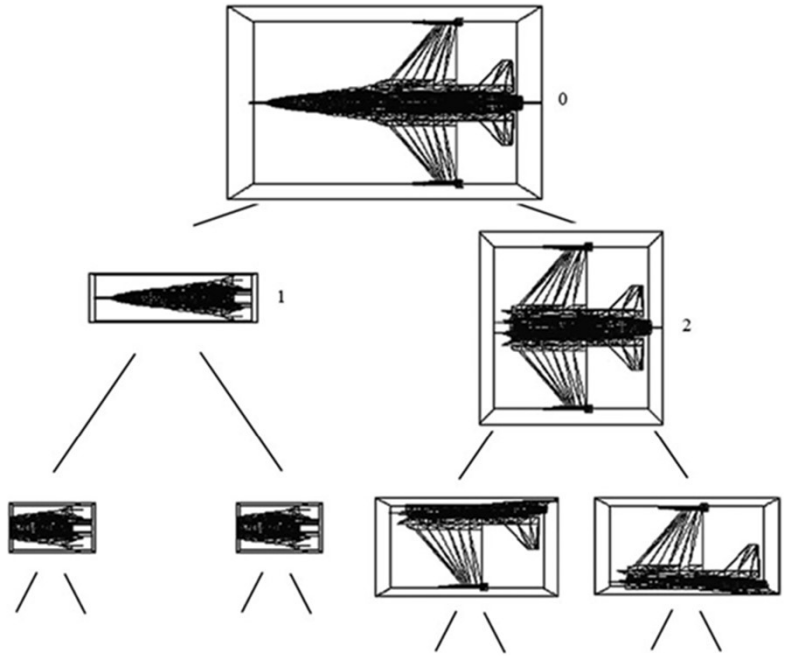
	Path tracing	BDPT	MMLT
Modern House	98.0%	97.6%	51.9%
San Miguel	95.9%	97.0%	62.0%

모듈 상세 설계 : 경로 추적

- 앙 MLT 함수띠

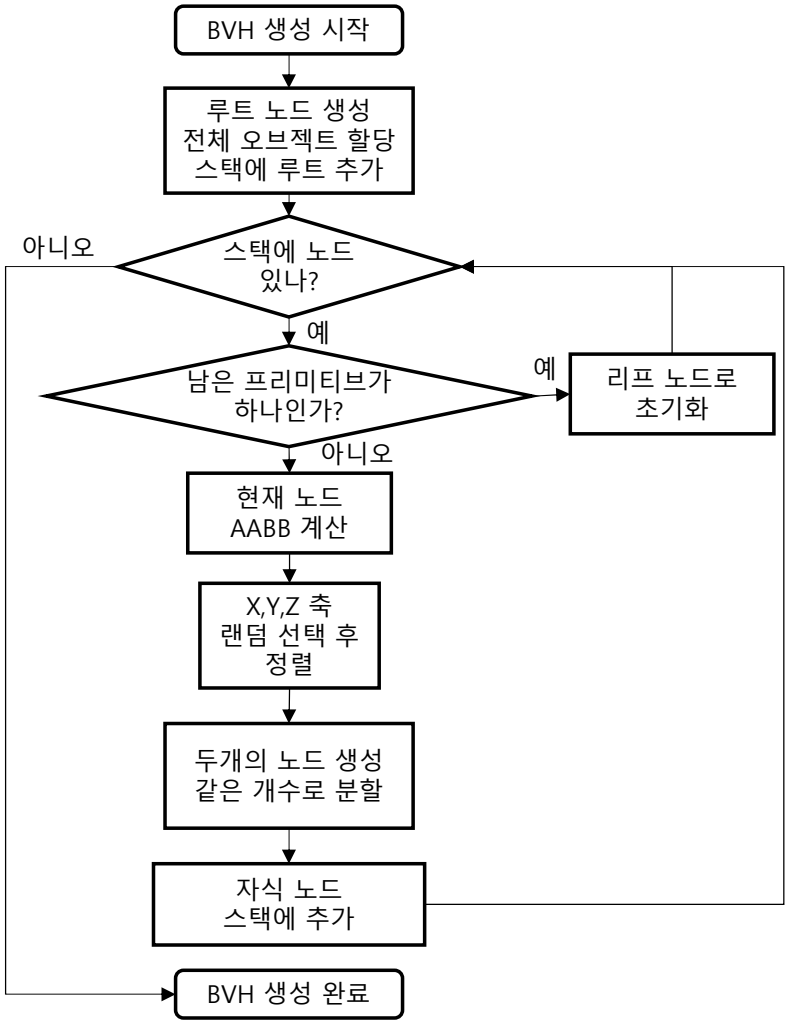
모듈 상세 설계 : BVH

- BVH: Bounding Volume Hierarchy
  - 이진 탐색 트리의 형태
- 씬 기하 탐색 : 교차 테스트
  - 선형/순차 탐색 : O(n)
  - BVH 탐색 : O(logn)
- 선형 탐색 사용시, 하나의 픽셀 계산 시에 최대 n * 25000번(500spp * 50depth)의 탐색
- 성능을 위하여 BVH 구현 필요



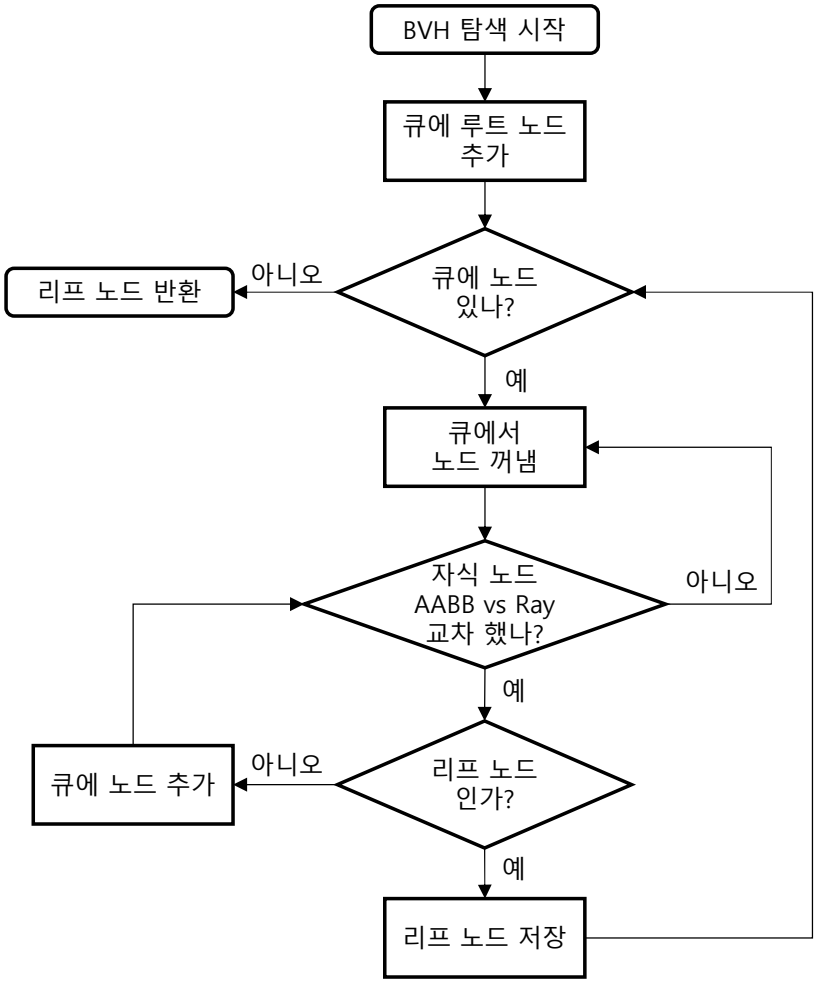
모듈 상세 설계 : BVH

- BVH 생성
  - 오브젝트 분할 방법 : 같은 개수의 오브젝트 할당
    - SAH 등 다른 방법이 존재



모듈 상세 설계 : BVH

- BVH 탐색
  - CUDA 상의 구현
    - 최대한의 적은 메모리 사용해 구현
    - 최적화 된 큐의 구현 필요



모듈 상세 설계 : BVH

메소드	**Intersect()**
형식	__device__ bool Intersect(IN const Ray& ray, OUT Intersection& isect);
반환 값	파라미터로 받은 ray와 부딪친 표면이 있으면 True, 아니면 False
설명	BVH 를 주어진 ray를 가지고 검사 후 탐색하여, 가장 가까운 물체의 충돌 정도를 isect 파라미터를 통해 넘김, 부딪친 Bounding Box를 가진 노드의 모든 자식들을 BFS 방식으로 큐를 가지고 검사하여 가장 가까운 물체를 찾음. Iaggregate 구현, GPU 에서 실행
예시	bool isIntersect = aggr.Intersect(currentRay, mIntersects[pixelIndex]);

Intersect() 데이터 구조		
데이터	자료형	설명
ray	const Ray&	부딪친 Primitive를 찾아내기 위해 사용되고, 이의 기준이 되는 광선
Isect	Intersection&(out)	최종적으로 가장 가까운 위치에서 부딪친 Primitive에 대한 정보를 해당 점에 대해 반환하는 파라미터

모듈 상세 설계 : BVH

메소드	**RecursiveBuild()**
형식	void RecursiveBuild(const FrameInput* input)
설명	BVH를 Top-Down 형태로 구성하는 함수, 각 내부 노드는 하위에 있을 모든 오브젝트의 Bounding Box 를 합쳐서 가지고 있고, 내부 노드가 가진 오브젝트가 2개 이하가 되기 전까지 아래 노드를 나누어 할당하고 Bounding Box 를 계산하는 것을 반복
예시	RecursiveBuild(&mRoot);

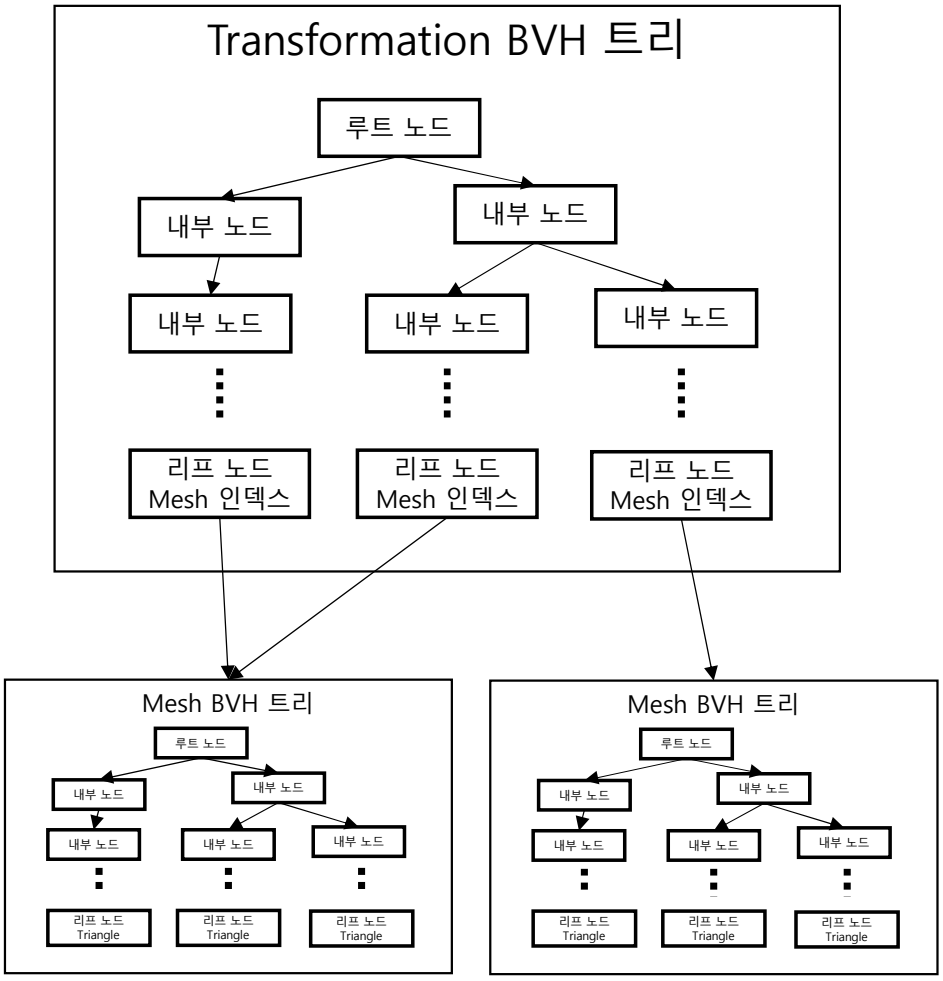
RecursiveBuild() 데이터 구조		
데이터	자료형	설명
input	const FrameInput*	해당 프레임의 데이터

메소드	**RecursiveBuildInternal()**
형식	void RecursiveBuildInternal(BVHNode** node)
설명	실질적인 BVH 생성을 해주는 함수
예시	RecursiveBuild(&mRoot);

RecursiveBuildInternal() 데이터 구조		
데이터	자료형	설명
node	BVHNode**	BVH의 내부 노드

모듈 상세 설계 : BVH

- BVH 생성 전략
  - 일반적인 생성 : 단일 BVH 트리
  - BVH 포레스트
    - Transformation BVH 트리
    - 각 Mesh 별로 BVH 트리
  - Skinned Mesh 가 아닌 경우, 여러 개의 프레임 계산 시에 Transformation BVH 트리만 재생성



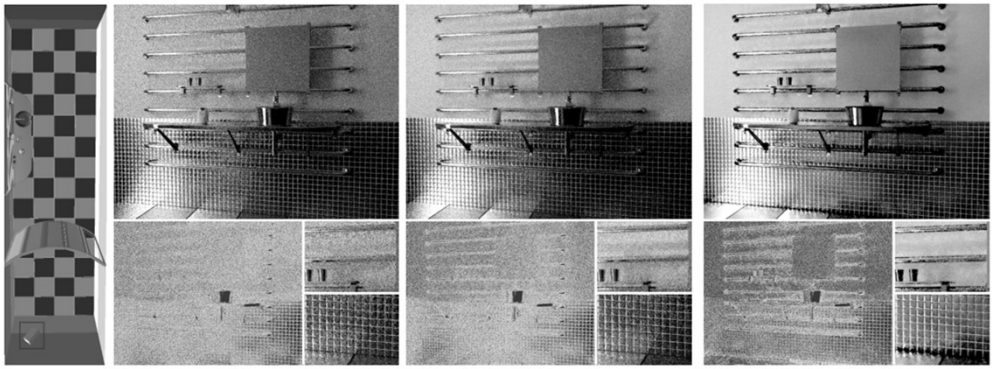
모듈 상세 설계 : BVH

메소드	**BuildBVHAs2Layer**
형식	void BuildBVHAs2Layer(IN const FrameInput* input, OUT BVH* bvhTransform, OUT BVH* staticMeshBVH, OUT int* staticMeshBVHCount, OUT BVH* dynamicMeshBVH, OUT int* dynamicMeshBVHCount);
설명	여러 개의 BVH를 생성하는 함수, 위에서 언급한 Transformation, Mesh 단위로 BVH를 나누어서 만듦. Dynamic BVH의 경우, 여러 개의 프레임을 생성 시, 다시 BVH를 생성해야 하므로 파라미터를 나눔
예시	BVH::BuildBVHAs2Layer(input, &bvhTransform, nullptr, &dynamicMeshBVH);

BuildBVHAs2Layer() 데이터 구조		
데이터	자료형	설명
input	const FrameInput*	해당 Frame의 데이터를 가지고 있는 마샬링된 데이터
bvhTransform	BVH*	Transform으로 만들어진 BVH
staticMeshBVH	BVH**	바뀌지 않는, 일반적인 Mesh로 BVH를 만듦. 배열의 포인터로 처리
staticMeshBVHCount	int*	Static Mesh BVH의 개수
dynamicMeshBVH	BVH**	Skinned Mesh 같은 변형되는 Mesh로 만들어질 BVH. 배열의 포인터로 처리
dynamicMeshBVHCount	Int*	Dynamic Mesh BVH의 개수

모듈 상세 설계 : 목표치 검증

- 실제와 비슷한, Ground Truth (reference image)와 만들어진 이미지를 IQA 기법을 사용하여 비교
  - Ground Truth는 Path-Tracing 기법으로 몇만번 이상의 샘플링을 사용하여 생성
    - Uniform random sampling
- 일반적으로 RMSE 사용
  - MS-SSIM, SC-QI 추천 [Whittle et al., 2017]
- MLT, PT 등 여러 기법에 대한 판단 방법이 될 수 있음



※ "목표치에 대한 검증 방법 완성도가 나와야함"

모듈 상세 설계 : 목표치 검증

메소드	**GetRMSE()**
형식	double GetRMSE(const ColorRGBA& image1, const ColorRGBA& image2);
반환 값	두 이미지 간의 RMSE 값
설명	RMSE 방법으로 두 이미지간의 에러 퀄리티를 계산하는 함수
예시	double err = GetRMSE(refImg, synthImg);

GetRMSE() 데이터 구조

데이터	자료형	설명
image1	const ColorRGBA&	비교할 이미지 1
image2	const ColorRGBA&	비교할 이미지 2

메소드	**GetMSSSIM()**
형식	double GetMSSSIM (const ColorRGBA& image1, const ColorRGBA& image2);
반환 값	두 이미지 간의 MS_SSIM 값
설명	MSE 계열의 MS-SSIM 방법으로 두 이미지간의 에러 퀄리티를 계산하는 함수
예시	double err = GetMSSSIM(refImg, synthImg);

GetMS_SSIM() 데이터 구조

데이터	자료형	설명
image1	const ColorRGBA&	비교할 이미지 1
image2	const ColorRGBA&	비교할 이미지 2

개발 환경

Windows 10

Visual Studio 2017.15.9.17

C++ / MSVC, CUDA / NVCC

NVidia CUDA Toolkit 10.1 Update 2

Unity 2019

C# / burst, MBE runtime

PC 1

CPU : Intel I7-4790

RAM: DDR3 16GB

GPU: GTX 970(4GB GDDR5)

PC 2

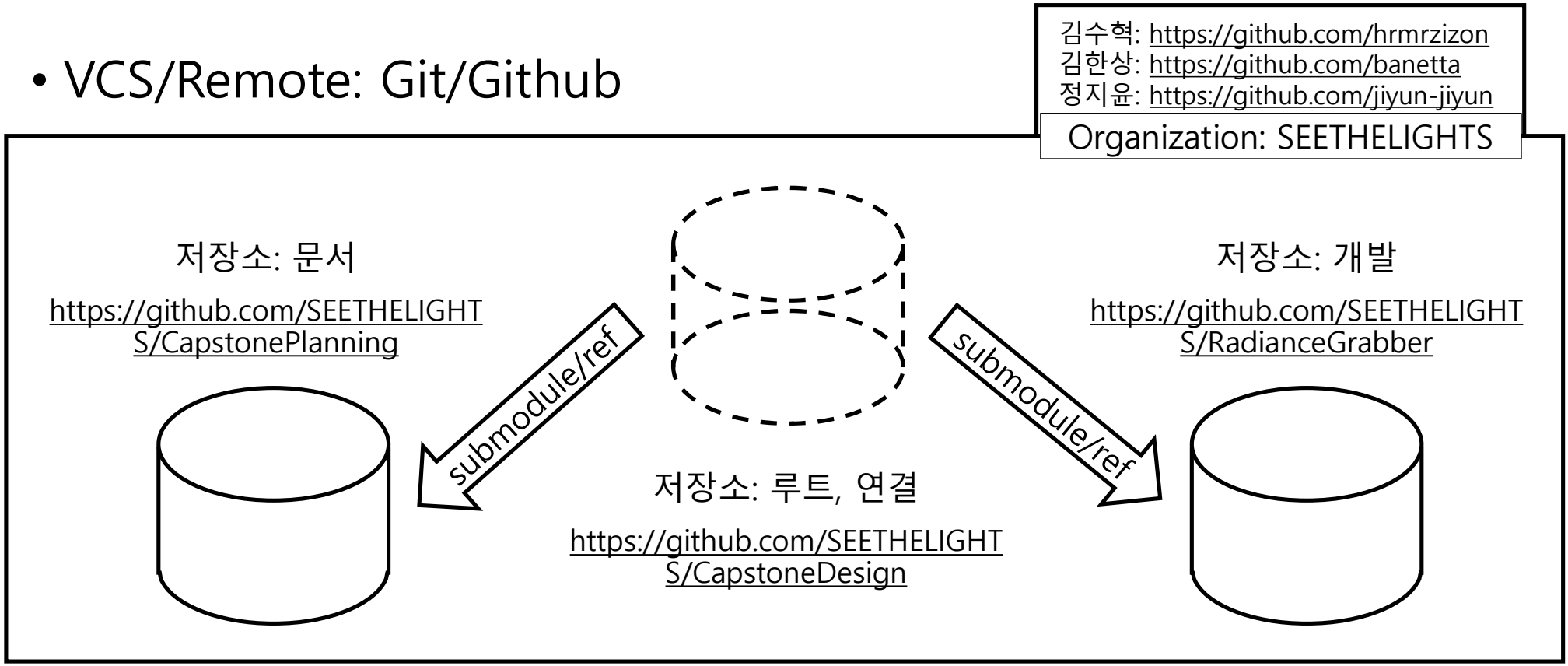
CPU: Intel I7-6700

RAM: DDR4 32GB

GPU: GTX-1070(8GB GDDR5)

개발 환경

- VCS/Remote: Git/Github



데모 환경 설계

- Unity 에디터에서 실행
- Unity 바닐라 Vs Ray Tracing
  - Unity 바닐라 렌더링 결과 vs 광선 추적 렌더링 결과
- 동영상 또한 동시에 비교하는 방식으로 구성
  - 미리 만들어진 동영상 재생

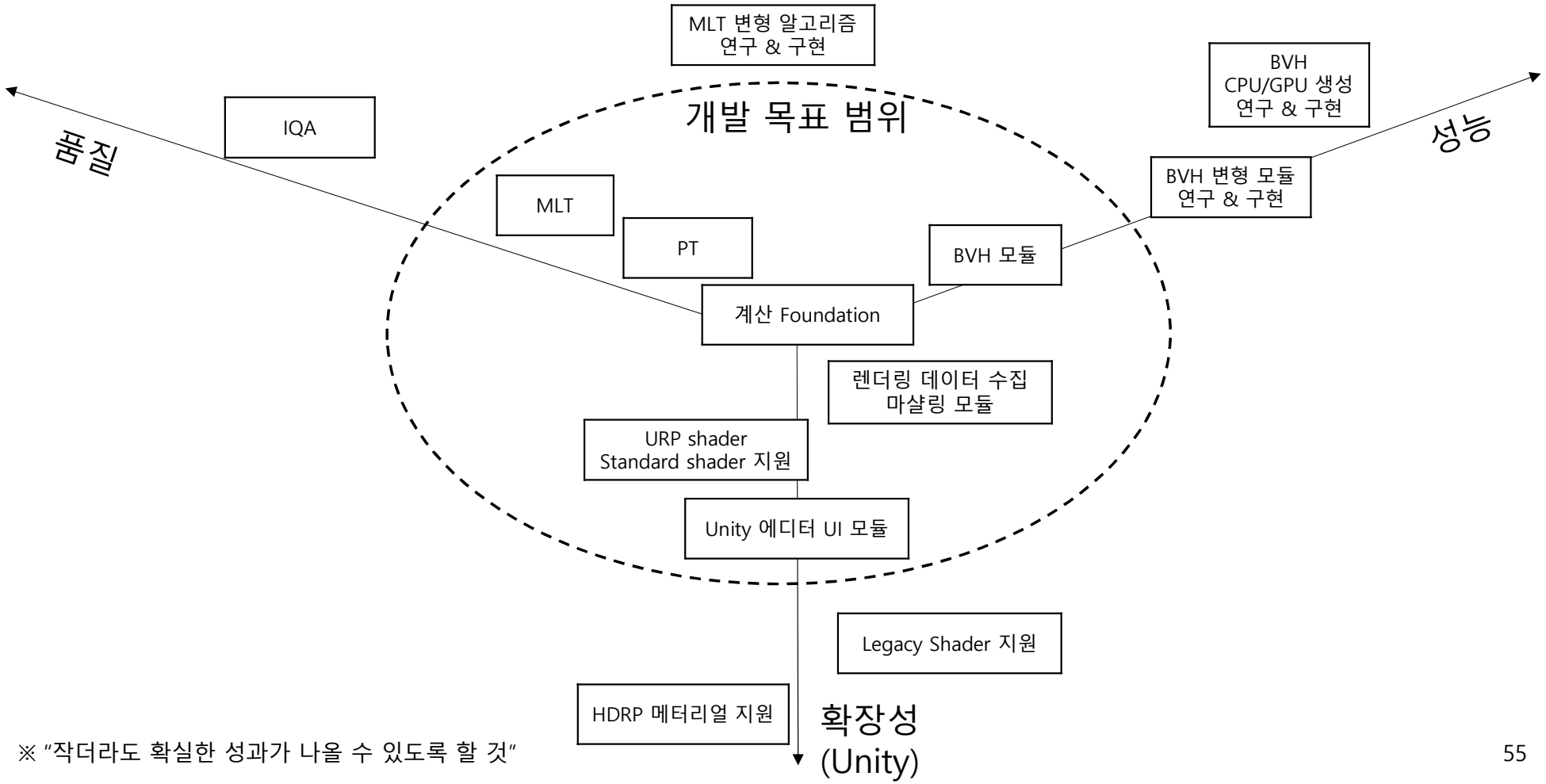
개발 방법

- 광선 추적기
  - 알고리즘 비교 연구 및 선택 (BVH, MLT, ..)
    - 공간 상의 물체들을 광선 추적시 빠르게 탐색 가능한 방법 비교 연구 필요
      - Bounding Volume Hierarchy, Linear BVH, SAH, ..
    - 보다 유효한 빛 줄기의 길을(high contribution path) 찾는 방법 비교 연구 필요 : Metropolis light transport, Bidirectional path tracing
      - noise, standard error 감소
    - Denoising 연구 필요 : cross bilateral filter, ..
  - 구현: C++/CUDA
    - 실질적인 계산 모듈: 렌더링 데이터를 GPGPU를 활용해 이미지의 픽셀의 색을 계산
    - CUDA를 사용하여 CPU/GPU 간의 메모리 동기화와 GPU에서의 프로그램을 실행
    - 넘겨받은 Graphics driver API 데이터를 직접 사용할 수 있도록 가공

개발 방법

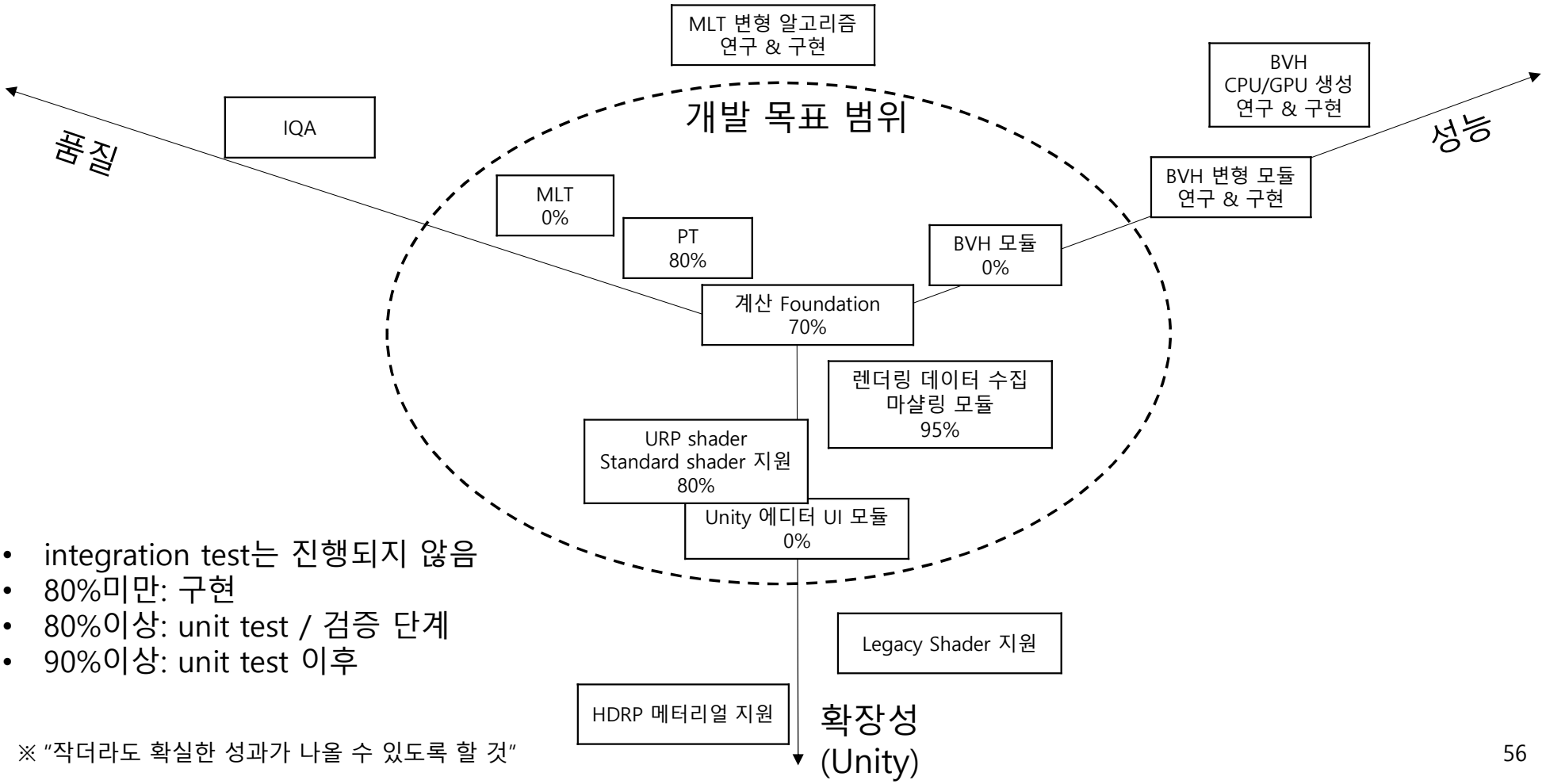
- Unity Plugin/Invoker(C#)
  - Unity 의 렌더링 데이터 수집: 각 프레임의 데이터, 프레임별 시간차
    - DirectX, OpenGL 등 Graphics driver API 에서 사용하는 데이터(vertex, index buffer, texture)
    - 각 표면들의 성질을 나타내는 Shader를 구현된 BSDF 중 하나에 맞게 매칭
    - 모든 그려지는 물체에 대한 변환 정보(위치,회전,크기)
  - 에디터 상의 결과 이미지 뷰어
  - 여러 개의 이미지를 통한 영상 녹화
    - 영상 녹화/코덱: 오픈소스 혹은 제공 기능 사용

개발 방법



※ "작더라도 확실한 성과가 나올 수 있도록 할 것"

구현 현황 : 60%



- integration test는 진행되지 않음
- 80%미만: 구현
- 80%이상: unit test / 검증 단계
- 90%이상: unit test 이후

※ "작더라도 확실한 성과가 나올 수 있도록 할 것"

업무 분담

	김수혁	김한상	정지윤
자료 수집	광선 추적, CUDA	CUDA	Unity 렌더링 환경
설계	광선 추적 계산 모델 설계 (알고리즘 선택 및 데이터 구조)	CUDA 기반의 Ray Tracing 기능 설계	Unity 환경 설계 (데이터 수집, 결과 출력)
구현	Foundation 구현(Geometry, Shader, etc..) CUDA 를 이용한 광선 추적 구현		Unity Plugin 의 형태로 렌더링 데이터 수집 및 결과 이미지 Intergration
테스트	테스트 (HW 별, 씬 복잡도-성능)		

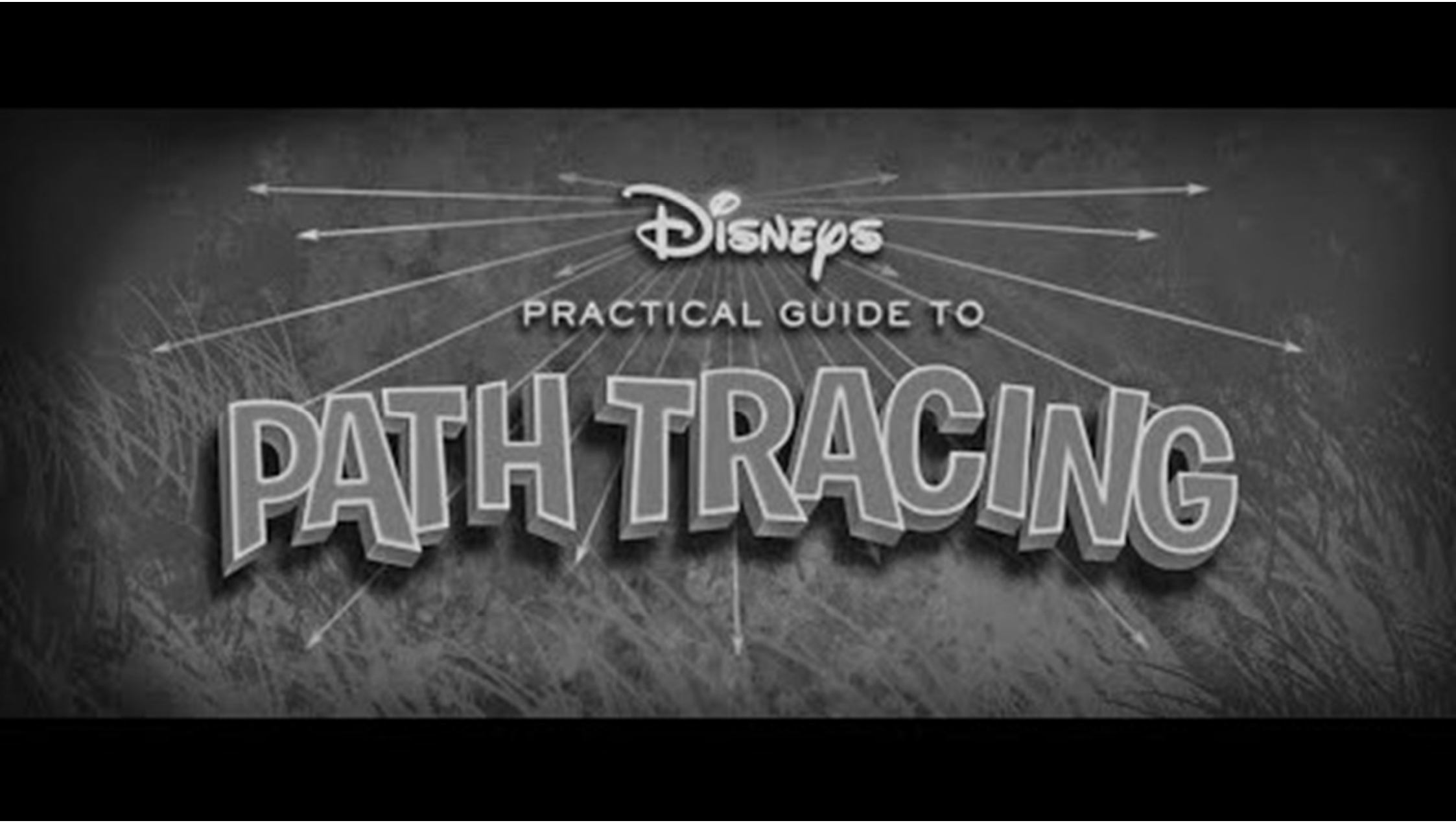
졸업 연구 일정

추진사항	12월	1월	2월	3월	4월	5월	6월	7-9월
자료조사 및 학습								
계산 모델 설계 SW 설계								
구현								
테스트 및 데모								
문서화 및 발표								
최종보고서 작성 및 발표								

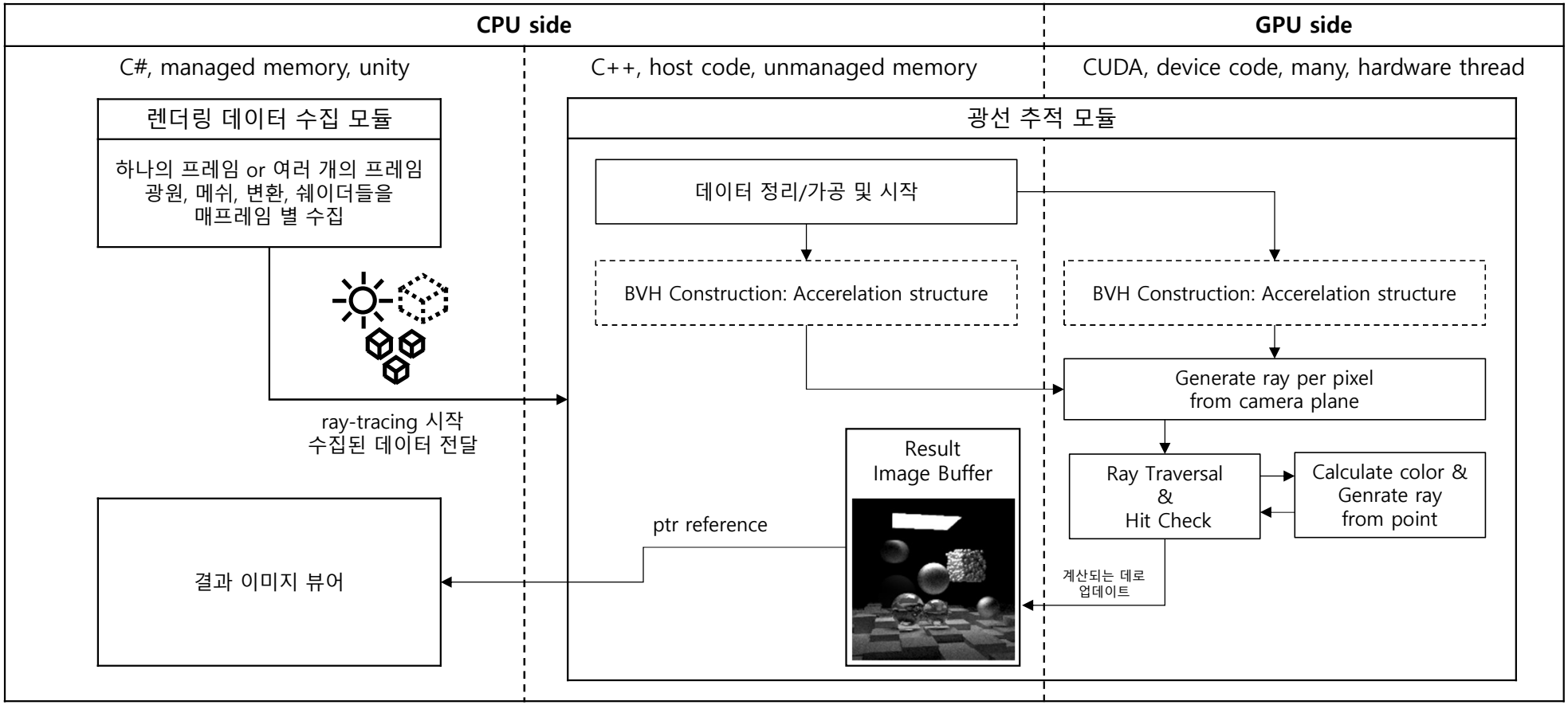
필요 기술 및 참고 문헌

- Unity
  - 이영진, 김중한, Unity Technologies: 게임 제작에도 족보가 있다, 삼성증권, 2019
  - Tianliang Ning, DirectX Ray Tracing in Unity 2019.3, Siggraph, 2019
  - Mono, Interop with Native Libraries, 2005
- CUDA
  - David B. Kirk, Wen-mei W.Hwu, 하순회(역), 김크리스(역), 이영민(역), 대규모 병렬 프로세서 프로그래밍, Bj 퍼블릭, 2010
  - Jason Sanders, Edward Kandrot, CUDA By Example, NVidia, 2010
  - Roger Allen, Accelerated Ray Tracing in One Weekend in CUDA, NVidia, 2018
- Ray tracing
  - Eric Haines (Editor), Tomas Akenine-Möller (Editor), Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs, 2019
  - Matt Pharr, Wenzel Jakob, Greg Humphreys, Physically Based Rendering: From Theory to Implementation, 2016
  - C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, D. Manocha, Fast BVH Construction on GPUs, 2009
  - Eric Veach, Leonida J. Guibas, Metropolis Light Transport, 1997
  - Eric P. Lafortune, Yves D. Willems, Bi-directional Path Tracing, 1993
  - Peter Shirley, Ray Tracing in One Weekend, 2018
  - Peter Shirley, Ray Tracing in The Next Weekend, 2018
  - Peter Shirley, Ray Tracing in The Rest of Your Life, 2018
  - Joss Whittle, Mark W. Jones, Rafał Mantiuk, Analysis of reported error in Monte Carlo rendered images, The Visual Computer, pp. 705-713, 2017
- Miscellaneous
  - Morgan McGuire, Computer Graphics Archive, July 2017 (https://casual-effects.com/data)

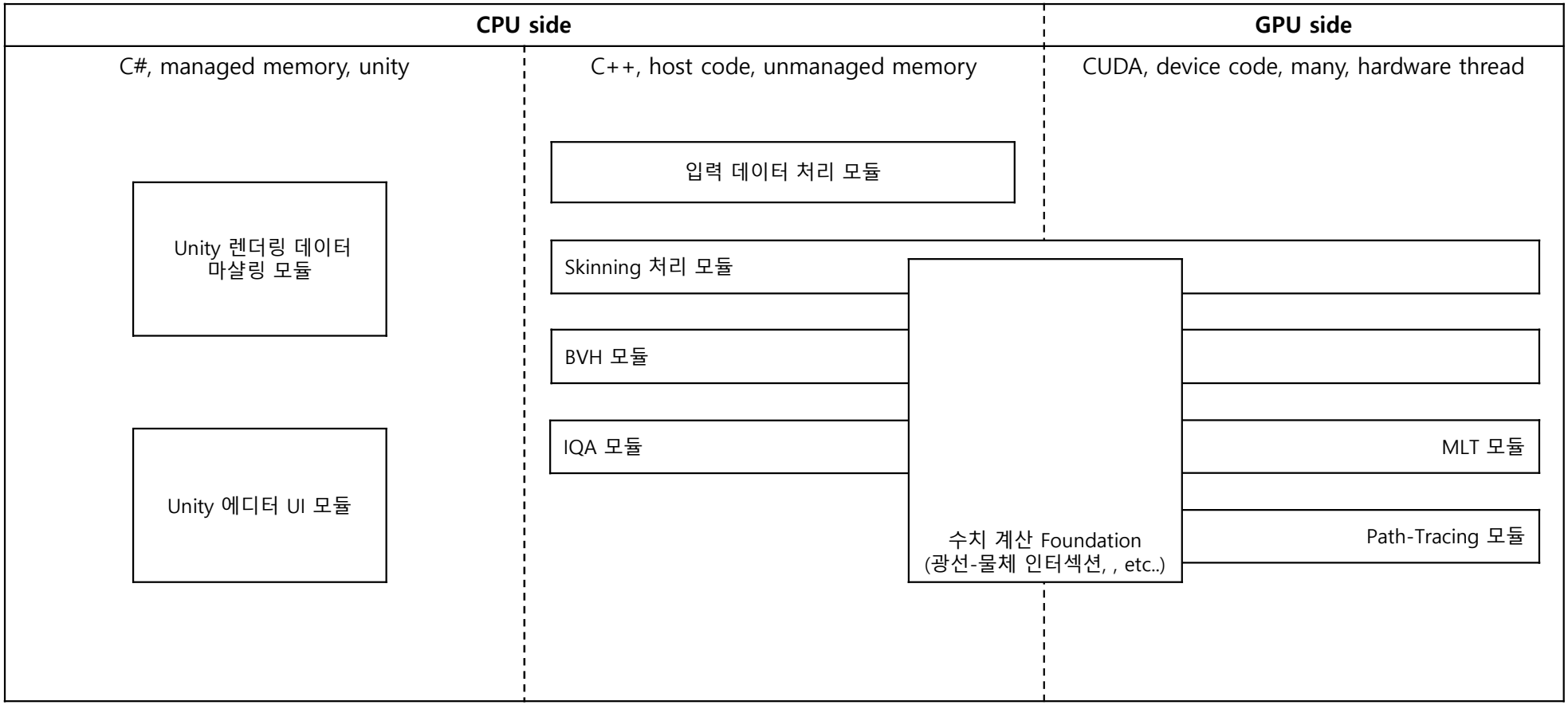
Thank you!


Disney's
PRACTICAL GUIDE TO
PATH TRACING

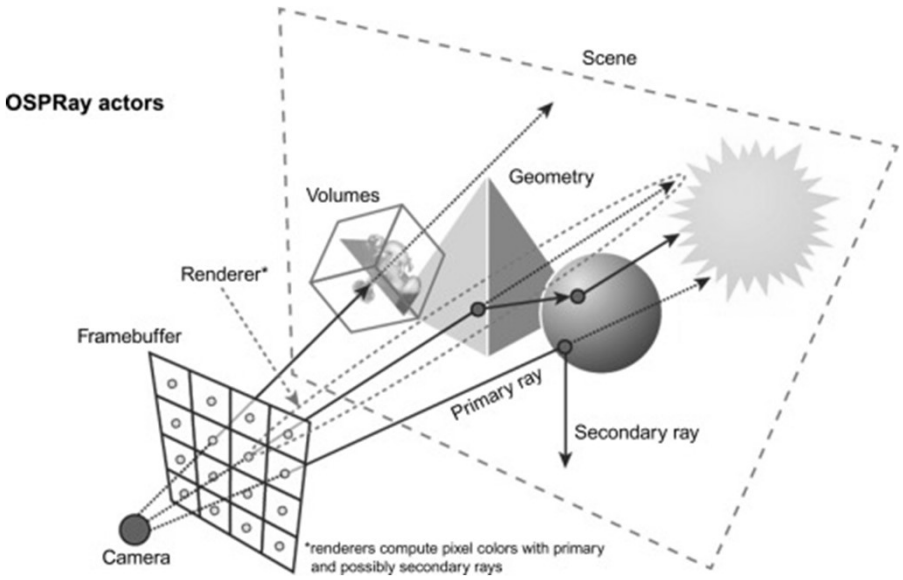
시스템 구성도


CPU side
GPU side
C#, managed memory, unity
C++, host code, unmanaged memory
CUDA, device code, many, hardware thread
렌더링 데이터 수집 모듈
하나의 프레임 or 여러 개의 프레임
광원, 메쉬, 변환, 쉐이더들을
매프레임 별 수집
ray-tracing 시작
수집된 데이터 전달
광선 추적 모듈
데이터 정리/가공 및 시작
BVH Construction: Accerelation structure
BVH Construction: Accerelation structure
Generate ray per pixel
from camera plane
Result
Image Buffer
Ray Traversal
&
Hit Check
Calculate color &
Genrate ray
from point
계산되는 데로
업데이트
ptr reference
결과 이미지 뷰어

모듈 상세 설계


CPU side
GPU side
C#, managed memory, unity
C++, host code, unmanaged memory
CUDA, device code, many, hardware thread
입력 데이터 처리 모듈
Unity 렌더링 데이터
마샬링 모듈
Skinning 처리 모듈
BVH 모듈
IQA 모듈
MLT 모듈
Unity 에디터 UI 모듈
Path-Tracing 모듈
수치 계산 Foundation
(광선-물체 인터섹션, , etc..)

모듈 상세 설계



Ray-Shape Intersection

- 수치 계산 Foundation
  - 벡터, 쿼터니온, 행렬 계산
  - 광선-물체 Intersection 계산
    - Ray vs Bouding Box, Triangle, ..

모듈 상세 설계 : 입력 데이터 마샬링

메소드	**StartSingleFrameGeneration**	클래스	**TaskController**
형식	int StartSingleFrameGeneration(FrameRequestOption opt, UnityRuntimeData runtimeData)		
반환 값	해당 Task의 ID		
설명	하나의 프레임 생성을 위한 DLL안의 함수 호출		
예시	int taskID = taskController.StartSingleFrameGeneration(opt, runtimeData);		

메소드	**StopSingleFrameGeneration**	클래스	**TaskController**
형식	public bool StopSingleFrameGeneration(int taskID)		
반환 값	찾아서 멈추었으면 True, 아니면 False		
설명	넘겨받은 task ID로 진행되던 이미지 생성을 중단		
예시	If (inputBuilder.ConvertRenderingData(currentTaskID)) Debug.LogFormat("이미지 생성({0})이 중지되었습니다.", currentTaskID);		

모듈 상세 설계 : 입력 데이터 마샬링

메소드	**StartMultiFrameRecord**	클래스	**TaskController**
형식	int StartMultiFrameRecord(MultiFrameRequestOption opt, UnityRuntimeData runtimeData)		
반환 값	해당 Task의 ID		
설명	여러 개의 프레임 생성을 위한 데이터 저장 준비		
예시	int taskID = taskController.StartMultiFrameRecord(mopt, runtimeData);		

메소드	**RecordOneFrame**	클래스	**TaskController**
형식	public void RecordOneFrame()		
설명	여러 개의 프레임 생성을 위한 현재 상태의 바뀔 수 있는 렌더링 정보를 저장		
예시	taskController.RecordOneFrame();		

메소드	**StopMultiFrameRecord**	클래스	**TaskController**
형식	public void StopMultiFrameRecord(int taskID)		
설명	여러 개의 프레임 생성 정보 저장 중지		
예시	taskController.StopMultiFrameRecord(selectedTaskID);		

모듈 상세 설계 : 입력 데이터 마샬링

메소드	**StartMultiFrameGeneration**	클래스	**TaskController**
형식	int StartMultiFrameGeneration()		
반환 값	해당 Task의 ID		
설명	여러 개의 프레임 생성을 위한 DLL안의 함수 호출		
예시	int taskID = taskController.StartMultiFrameGeneration();		

메소드	**StopMultiFrameGeneration**	클래스	**TaskController**
형식	public bool StopMultiFrameGeneration(int taskID)		
반환 값	찾아서 멈추었으면 True, 아니면 False		
설명	여러 개의 프레임 생성 취소를 위한 DLL안의 함수 호출		
예시	If (taskController.StopMultiFrameGeneration(selectedTaskID);) Debug.LogFormat("이미지 생성({0})이 중지되었습니다.", selectedTaskID);		

모듈 상세 설계 : 경로 추적

메소드	**Render**	클래스	**PathIntegerator**
형식	void Render(const IAggregate& scene)		
설명	주어진 정보를 가지고 Path-Tracing 을 수행, IIntergrator 구현		
예시	path.Render(bvh);		

메소드	**GenerateRay**	클래스	**PathIntegerator**
형식	__global__ void GenerateRay(PathSegment* semgents)		
설명	카메라를 기준으로 광선 방향과 위치 설정, GPU 커널 실행 함수		
예시	GenerateRay<<<optimalGridDim, optimalBlockDim>>>(mSegments);		

메소드	**ScatteringAndAccumAttenuation**	클래스	**PathIntegerator**
형식	__global__ void ScatteringAndAccumAttenuation(PathSegment* semgents, Intersection* isects)		
설명	IAggregate 객체를 가지고 반환한 표면에 대한 정보를 통해 광선 방향과 감쇠 계수 설정, GPU 커널 실행 함수		
예시	ScatteringAndAccumAttenuation<<<optimalGridDim, optimalBlockDim>>>(mSegments, mIsects);		

모듈 상세 설계 : 경로 추적

메소드	**Intersect**	클래스	**LinearAggregation**
형식	__device__ bool Intersect(IN const Ray& ray, OUT Intersection& isect);		
반환 값	파라미터로 받은 ray와 부딪친 표면이 있으면 True, 아니면 False		
설명	선형/순차적으로 모든 물체들과의 Ray vs BoundingBox 검사를 실시함. IAggregate 구현, GPU 에서 실행		
예시	bool isIntersect = aggr.Intersect(currentRay, mIntersects[pixelIndex]);		

메소드	**RecursiveBuild**	클래스	**LinearAggregation**
형식	void RecursiveBuild(BVHNode** node)		
설명	BVH를 Top-Down 형태로 구성하는 함수, 각 내부 노드는 하위에 있을 모든 오브젝트의 Bounding Box 를 합쳐서 가지고 있고, 내부 노드가 가진 오브젝트가 2개 이하가 되기 전까지 아래 노드를 나누어 할당하고 Bounding Box 를 계산하는 것을 반복한다.		
예시	RecursiveBuild(&mRoot);		

모듈 상세 설계 : 경로 추적

메소드	**Intersect**	클래스	**BVH**
형식	__device__ bool Intersect(IN const Ray& ray, OUT Intersection& isect);		
반환 값	파라미터로 받은 ray와 부딪친 표면이 있으면 True, 아니면 False		
설명	BVH 를 주어진 ray를 가지고 검사 후 탐색하여, 가장 가까운 물체의 충돌 정도를 isect 파라미터를 통해 넘김, 부딪친 Bounding Box를 가진 노드의 모든 자식들을 BFS 방식으로 큐를 가지고 검사하여 가장 가까운 물체를 찾음. Iaggregate 구현, GPU 에서 실행		
예시	bool isIntersect = aggr.Intersect(currentRay, mIntersects[pixelIndex]);		

메소드	**RecursiveBuild**	클래스	**BVH**
형식	void RecursiveBuild(BVHNode** node)		
설명	BVH를 Top-Down 형태로 구성하는 함수, 각 내부 노드는 하위에 있을 모든 오브젝트의 Bounding Box 를 합쳐서 가지고 있고, 내부 노드가 가진 오브젝트가 2개 이하가 되기 전까지 아래 노드를 나누어 할당하고 Bounding Box 를 계산하는 것을 반복		
예시	RecursiveBuild(&mRoot);		

모듈 상세 설계

- Skinning
- 1. 기존의 메쉬 구조 ➔ 스킨드 메쉬 구조 이유?
- 2.