INVEN

igc in 성남

검은사막의 렌더링

최창애 (choichangae@pearlabyss.com)
펄어비스

INVEN GAME CONFERENCE IN SEONGNAM

igc in 성남

INDEX

igc in 성남

자기소개

- 저는

평범한 주부, 아이 엄마이자 프로그래머입니다.

전) 웹젠 C9엔진팀

현) 펄어비스 프로그램 1팀 팀장

검은사막에서 렌더링, 레벨 에디터, 리마스터링 작업 중

- 펄어비스 그래픽스 엔진 관련 프로그래머
- 3개팀 김대일 의장님 포함 팀장3명, 총 15명 + α

  렌더링, 최적화 작업

  - ✓ Black Desert Online(PC) – D3D9, D3D11 개발, 유지보수
  - ✓ Black Desert Mobile – OpenGL ES 3.1 지원
  - ✓ Black Desert XB1 - D3D11 개발 중
  - ✓ 신작 PS4 – GNM 개발 중

1

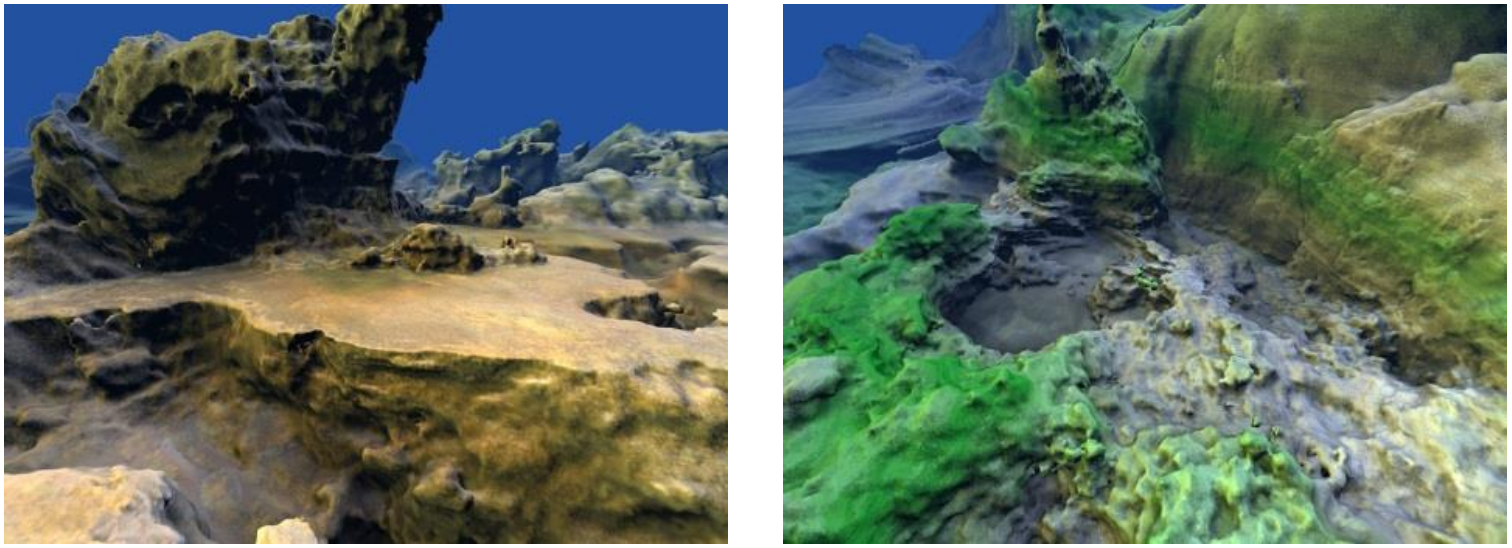
검은사막의 렌더링

igc in 성남

복셀 기반 폴리곤 지형

- **Voxel : volumetric element**

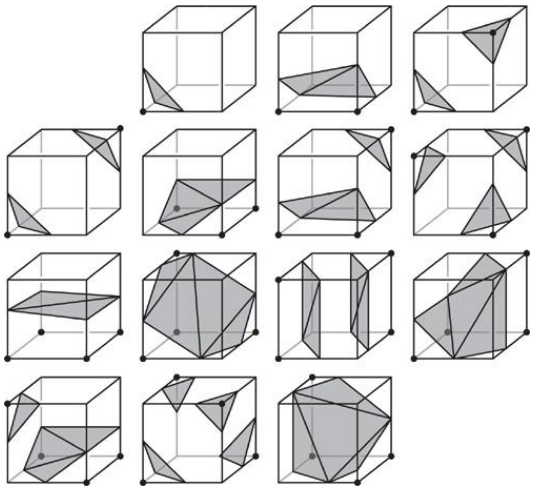
- **The density function[1] :** 3D 공간의 모든 점에 대해
  - ✓ 값이 양수이면 지형 표면 안에 속함
  - ✓ 값이 음수이면 빈 공간 (예 : 대기 또는 물)에 속함
  - ✓ 밀도 값이 0 인 경계는 지형의 표면
  - ✓ 복셀 꼭지점의 밀도값을 RLE(Run-length encoding)로 저장



Voxel Terrain

복셀 기반 폴리곤 지형

- **Marching cube[2] :** 각각의 복셀의 8개의 밀도값으로 폴리곤을 생성.
  - ✓ 밀도가 모두 같은 부호이면 지형 내부 또는 외부에 있으므로 폴리곤은 출력되지 않음
  - ✓ 그밖에 경우 지형과 빈 공간 경계에 놓여 있어서 폴리곤이 생성됨
  - ✓ 2개의 밀도값을 보간하여 변 위에 한 점을 지정하고 각 점 3개를 연결하여 폴리곤 생성



Fundamental Cases for Marching Cubes

- **Pre-computation** : 레벨 에디터에서 복셀기반 에디팅 후 폴리곤 메쉬를 미리 생성
  런타임에는 폴리곤 메쉬만 사용

복셀 기반 폴리곤 지형

- **Pros[3]:**
  - ✓ **Continuous 3D data** : 숨겨진 동굴 같은 지형에 대한 연속 데이터를 저장하는 효율적인 방법
  - ✓ **Easy to modify** : 쉽게 수정 가능
  - ✓ **Very fast** : GPU Voxel Terrain에 비해 GS가 필요하지 않음
  - ✓ **No artifacts** : 메쉬가 있는 그대로 렌더링되므로 이상한 경계나 glitch가 발생하지 않음
  - ✓ **Advanced terrain features** : 구멍을 뚫거나 돌출부를 만들고 seamless 터널을 자연스럽게 만들 수 있음
- **Cons:**
  - ✓ **Poor dynamic LOD** : 메쉬 LOD방식 사용
  - ✓ collision detection 또한 일반 메쉬 방식. 지형 높이를 얻기 위해 컬리전 활용



복셀 기반 폴리곤 지형 영상

Deferred Shading

- **전통적인 deferred shading 방식[4]**
  1. BRDF의 평가에 필요한 정보를 **geometry buffer(G-Buffer)**에 렌더링
     - ✓ 재질속성(albedo, material info.), 표면 속성(normal)
       표면속성(positon)은 depth로 부터
     - ✓ 불투명 재질의 객체는 그림자 패스를 빼면 한번만 렌더
     - ✓ G-Buffer 렌더링 시 조명 정보가 필요하지 않으므로, 일괄 단위로 함께 렌더링 할 수 있는 **인스턴스**의 수가 늘어남
     - ✓ MSAA 부담스러워 **FXAA**사용
  2. 그림자 렌더
  3. 각각 개별적인 쉐이더를 이용해 조명, AO등 Screen Space에서 계산
  4. 최종 결과를 결합
  5. 간단한 조명 계산의 투명 재질, 헤어 렌더
  6. Post-processing

Deferred Shading

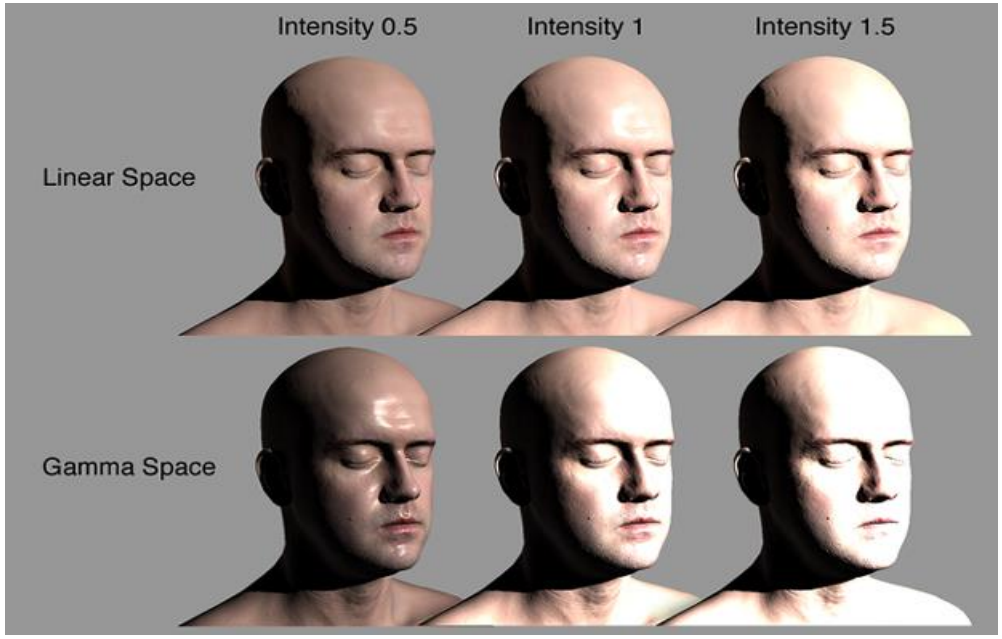
- **G-buffer : 메모리와 대역폭 줄이고, 재질마다 다른 BRDF사용하기 위해 노력**
  1. **RT0**
     - ✓ **R : encoded Diffuse color의 Y'**
     - ✓ **G : encoded Diffuse color의 Cb [half] / Cr [half]**
     - ✓ **B : encoded Vertex Normal.x [half] / y [half]**
     - ✓ **A : Material ID**
  2. **RT1**
     - ✓ **R : encoded Normal.x**
     - ✓ **G : encoded Normal.y**
     - ✓ **B : Gloss(1-roughness) [half] / Fresnel[half]**
     - ✓ **A : Emissive or Anisotropy [half] / Skin Mask [half]**
  3. **Depth Buffer**
     - ✓ **Position**
- [half]는 interlaced, 홀수와 짝수 세로줄로 나누어 번갈아가며 저장
- Diffuse encoding[5] : YUV encoding, 인간의 눈은 밝기의 변화보다는 색조의 변화에 덜 민감, Luma는 full size로 Chroma는 half size로 저장
- Normal encoding[6] : x,y 평면으로 z 벡터를 투영하여 flaot2로 인코딩

PBR 기반 작업

- **Why Physically based?**
  - ✓ 좀 더 쉽게 사실적인 이미지를 얻기 위해서
  - ✓ 아티스트를 위한 간단한 Material Interface
    - 몇 가지 정해진 매개변수 입력으로 물리적인 표현 가능
    - 더 이상 아티스트가 감으로 작업하지 않아도 됨
  - ✓ 누가 작업하더라도 일관되게 할 수 있음.

PBR 기반 작업

- **효과적인 PBR을 위한 기반 작업**
  - ✓ **Gamma-Correct Rendering[7]**
    - 모니터 색 공간 비선형
    - 비선형 gamma space에서의 shading 연산은 잘못된 결과를 초래
    - nonlinear encoding된 텍스처 입력을 linear space로 변환 후 shading
    - 최종 출력은 다시 nonlinear encoding
  - ✓ **HDR**
  - ✓ **HDR environment map**



Linear Space vs Gamma space

HDR(A2R10G10B10)[8]

- **Dynamic Range :** 표현할 수 있는 가장 큰 값과 작은 값의 비율
- real-world의 Dynamic Range는 100,000 : 1일 수 있음

1. *Frame buffer of absolute luminance*
   - ✓ *실제 luminance space로 렌더링*
   - ✓ 2^{32}이면 모든 범위 luminance를 직접 표현 가능
2. Displayable image
   - ✓ 2^{8} Low dynamic range (LDR)
3. Frame buffer of relative luminance
   - ✓ 렌더링 시 노출 값에 대한 스케일 적용
   - ✓ 이상적으로, $2^{15\sim16}$ 이상
   - ✓ 게임에서는 $2^{12\sim13(4000\sim10000)}$ 정도면 적당

- **검은사막 : A2R10G10B10,** 적은 메모리 소비
  - ✓ 리마스터링에서 R16G16B16A16로 수정
- Low-precision environment maps
  - ✓ 마찬가지로 리마스터링에서 수정

- **Physically Based Rendering[9][10][11]**
  - ✓ **Diffuse BRDF : Oren-Nayar[12]**
    - rough diffuse surface을 설명하는 인기 모델
    - 램버트 반사에 비해 일반적으로 거친 표면에서 조명 방향이 시야 방향에 다가갈수록 더 밝아짐
    - 램버트 반사에 비해 외곽 경계 반사, 명암 경계의 전환이 부드러운 것을 볼 수 있음

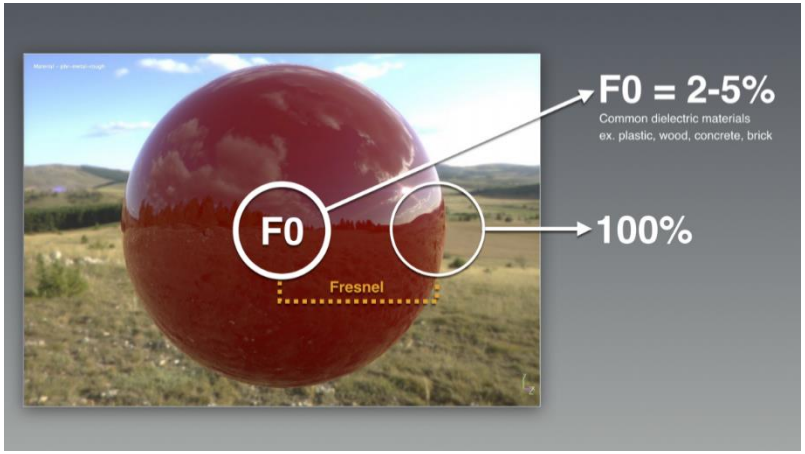
Lambert Diffuse vs Oren-Nayar Diffuse
(http://www1.cs.columbia.edu/CAVE/publications/pdfs/Oren_SIGGRAPH94.pdf)

- **Physically Based Rendering**
  - ✓ **Microfacet Specular BRDF :**

$$f(\mathrm{l}, \mathrm{v}) = \frac{D(\mathrm{h})\, F(\mathrm{l}, \mathrm{h}) G(\mathrm{l}, \mathrm{v}, \mathrm{h})}{4(\mathrm{n} \cdot \mathrm{l})(\mathrm{n} \cdot \mathrm{v})}$$

1. Fresnel [13]

- $f(\mathrm{l}, \mathrm{v}) = \frac{D(\mathrm{h})\, \boldsymbol{F(\mathrm{l}, \mathrm{h})} G(\mathrm{l}, \mathrm{v}, \mathrm{h})}{4(\mathrm{n} \cdot \mathrm{l})(\mathrm{n} \cdot \mathrm{v})}$
- 입사각에 따른 반사와 투과에 대한 함수
- 표면에서 반사되는 빛의 양은 시야각에 달려 있음.
- 수평 입사각(90도 입사각)에서 물처럼 매끄러운 표면은 거의 100%반사, 수직 입사각(0도 입사각)에서의 반사량은 재질의 굴절률에 따라



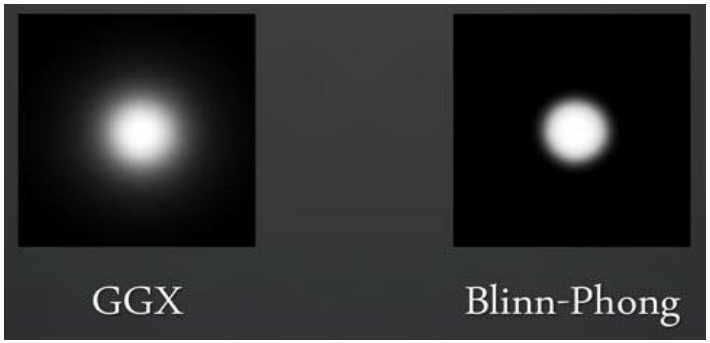
1. Fresnel(cont.)

- **The shlick Approximation**[13] 사용(상당히 정확하고, 값싼 근사)
- $F(v, h) = F_0 + (1 - F_0)\, 2\,(-5.55473(v \cdot h) - 6.98316)(v \cdot h)$
- **F_0** 를 매개변수로 입력.
- $f(0^o) = \frac{(n-1)^2}{(n+1)^2}$, n은 굴절률 (Index of Refraction , IOR)
- 일반적인 유전체의 **F_0** 는 0.02~0.05범위로 그레이 스케일
- 도체의 경우 0.5~1.0범위로 RGB값을 가짐

Material	F_0 (Linear)	F_0 (sRGB)	Color
Water	(0.02, 0.02, 0.02)	(0.15, 0.15, 0.15)	
Plastic / Glass (Low)	(0.03, 0.03, 0.03)	(0.21, 0.21, 0.21)	
Plastic High	(0.05, 0.05, 0.05)	(0.24, 0.24, 0.24)	
Glass (high) / Ruby	(0.08, 0.08, 0.08)	(0.31, 0.31, 0.31)	
Diamond	(0.17, 0.17, 0.17)	(0.45, 0.45, 0.45)	
Iron	(0.56, 0.57, 0.58)	(0.77, 0.78, 0.78)	
Copper	(0.95, 0.64, 0.54)	(0.98, 0.82, 0.76)	
Gold	(1.00, 0.71, 0.29)	(1.00, 0.86, 0.57)	
Aluminium	(0.91, 0.92, 0.92)	(0.96, 0.96, 0.97)	
Silver	(0.95, 0.93, 0.88)	(0.98, 0.97, 0.95)	

2. Normal Distribution Function

- $f(l, v) = \frac{\boldsymbol{D(h)}\, F(l, h)\, G(l, v, h)}{4(n \cdot l)(n \cdot v)}$
- NDF는 하이라이트의 크기와 모양 결정
- **Trowbridge-Reitz (GGX)**[15] 사용
- $D(h) = \frac{\alpha^2}{\pi((n \cdot h)2(\alpha^2 - 1) + 1)^2}$
- **α** is **roughness**



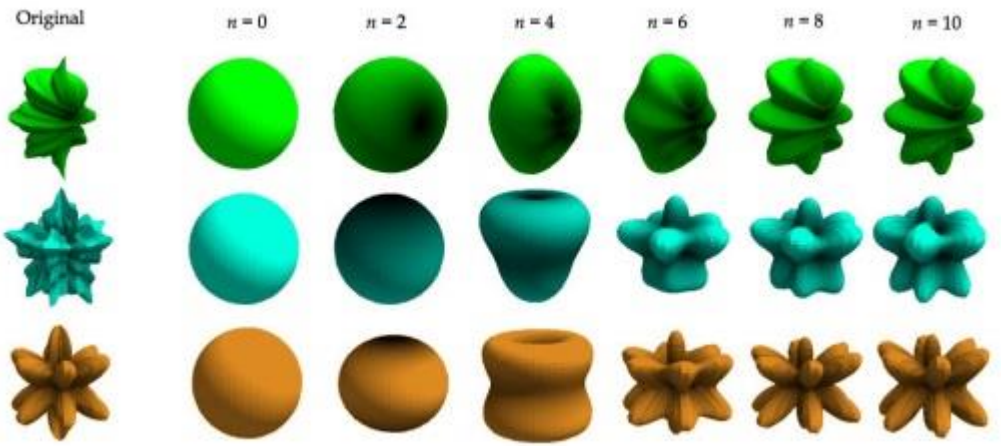
GGX vs Blinn-Phong

3. Geometric Shadowing

- $f(l, v) = \frac{D(h)\, F(l, h)\, \boldsymbol{G(l, v, h)}}{4(n \cdot l)(n \cdot v)}$
- Microfacet에서 일부 out-going beam 블럭됨(masking), 일부 incoming beam 블럭됨(self-shadowing)
- **Cook-Torrance**[16][17] 사용
- $Gct(l, v, h) = \min(1, \frac{2(n \cdot h)(n \cdot v)}{(v \cdot h)}(G_{masking}), \frac{2(n \cdot h)(n \cdot l)}{(v \cdot h)}(G_{shadowing}))$

SH Lighting

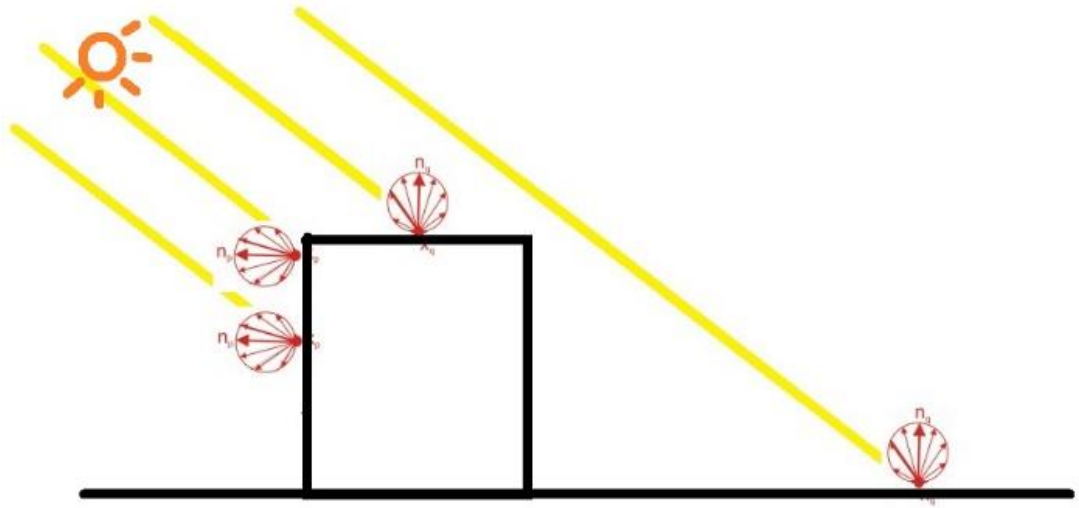
- Point Lights / Spot Lights / GI Virtual Point Lights(VPLs) **무한** 렌더
- **Spherical Harmonics**[18]를 이용하여 라이팅 연산 결과를 합함
- SH 계수만 저장하여 압축(Low memory consumption)
- **2차**(second order), 4개 SH 계수(coefficient) * r, g, b 채널 * ¼ Size RT
- Diffuse Lighting에 3차로 저장하면 충분, 메모리/성능 관계상 2차 사용
- 복원된 근사 mass light direction으로 specular 연산 까지!
- Full size 조명 연산보다는 밝기, 색감 차이 존재
- GI VPL와 거리가 먼 조명에 쓰임



SH projection of functions with increasing orders of approximation
(http://silviojemma.com/public/papers/lighting/spherical-harmonic-lighting.pdf)

Effects of Global Illumination Approximation

- Directional Light Source로 부터 CPU에서 **ray-casting**
- 교차되는 표면에 격자로 **VPLs** 생성
  - ✓ 교차 표면 position(a), normal(b) 저장
- **Shadow Map** 렌더 시 Shadowmap Space의 Diffuse(albedo) Color 준비
- Screen Space에서 Diffuse Color * shadow값 저장(c)
- **SH Lighting**에서 (a), (b), (c)값으로 Lighting 연산



Effects of GI Approximation

Effects of Global Illumination Approximation

- GI근사 효과 적용 전

Effects of Global Illumination Approximation

- GI근사 효과 적용 후

Rendering Huge World

- 검은사막 전체 월드 크기(현재)
  : 34.816km x 31.744km (x좌표 가장 큰수 대략 1,420,800)
- 원점에서 멀어 질수록 부동 소수점 정밀도 문제 생김
- 그럼 어떻게 해결?
  - ✓ use View Relative Position

```
matModelVRP._41 -= viewPos.x;
matModelVRP._42 -= viewPos.y;
matModelVRP._43 -= viewPos.z;

matViewOrigin
        loc : float3::Zero
        lookAt : viewDirection
        Up : float3:: UnitY

matModelVRP * matViewOrigin * matProj
```

검은사막 월드

최적화

- **Instancing**
  - ✓ 모든 geometry는 instancing으로 렌더
- **Rendering Order**
  - ✓ instancing 데이터 구성 시 텍스처로 먼저 소팅 후 메쉬 데이터로 소팅
  - ✓ texture 바인딩을 vertex buffer 바인딩보다 적게!
- **Command Buffer**
  - ✓ **Dx11의 deferred context개념**
  - ✓ graphics commands를 command buffer에 레코딩 후
  - ✓ render thread에서 play back하는 방식
  - ✓ 검은사막 DX9, DX11에서 활용, OpenGL ES(모바일), 콘솔에서도 사용 중!
- **Vertex Texture Fetch(VTF)**
  - ✓ 캐릭터 GPU Skinning, 일부 instancing data에 활용

최적화

- **Crop Mode**
  - ✓ 화면의 가장자리 부분은 UI만 렌더링, 게임 화면은 해상도보다 작은 사이즈로 렌더

- **Upscaling**
  - ✓ 게임 화면을 해상도보다 작은 사이즈로 렌더링 후 Bicubic Upsizing[19]

Bicubic upsizing

2

레벨 에디터, 생산성 향상을 위한 노력들

igc in 성남

레벨 에디터 개발의 방향

아티스트와 함께 필요한 순서로 하나씩 기능 추가

- 작업자가 **바로 바로 확인** 할 수 있게
  - ✓ pre-computation 작업 최소화
- **여러 명이 동시**에 작업이 가능하게
  - ✓ 레벨 데이터를 지역별, 분류별(지형, 오브젝트, 나무 등) 작은 단위로 나누어 리비전 관리, 배포시에 병합
  - ✓ 라이브 / 업데이트 데이터 나누어 동시 작업
- **프로토타입을 빠르게** 만들어 확인 후 세부 작업이 가능하게
  - ✓ hightmap으로부터 복셀 데이터 importing
  - ✓ 레벨 오브젝트 그룹 배치
  - ✓ 그룹 템플릿
  - ✓ decal volume 기능 등
- 최대한 자동화
  - ✓ 네비 데이터, 충돌 데이터, LOD, road decal, ivy vine
- **최적화는 리소스가 아닌 코드로!**

레벨 오브젝트 그룹 배치

- 허허벌판

레벨 오브젝트 그룹 배치

- 레벨 디자이너가 스무 번 미만 클릭

레벨 오브젝트 그룹 템플릿

- 그룹 템플릿 객체 여섯 개 배치

Decal Volume

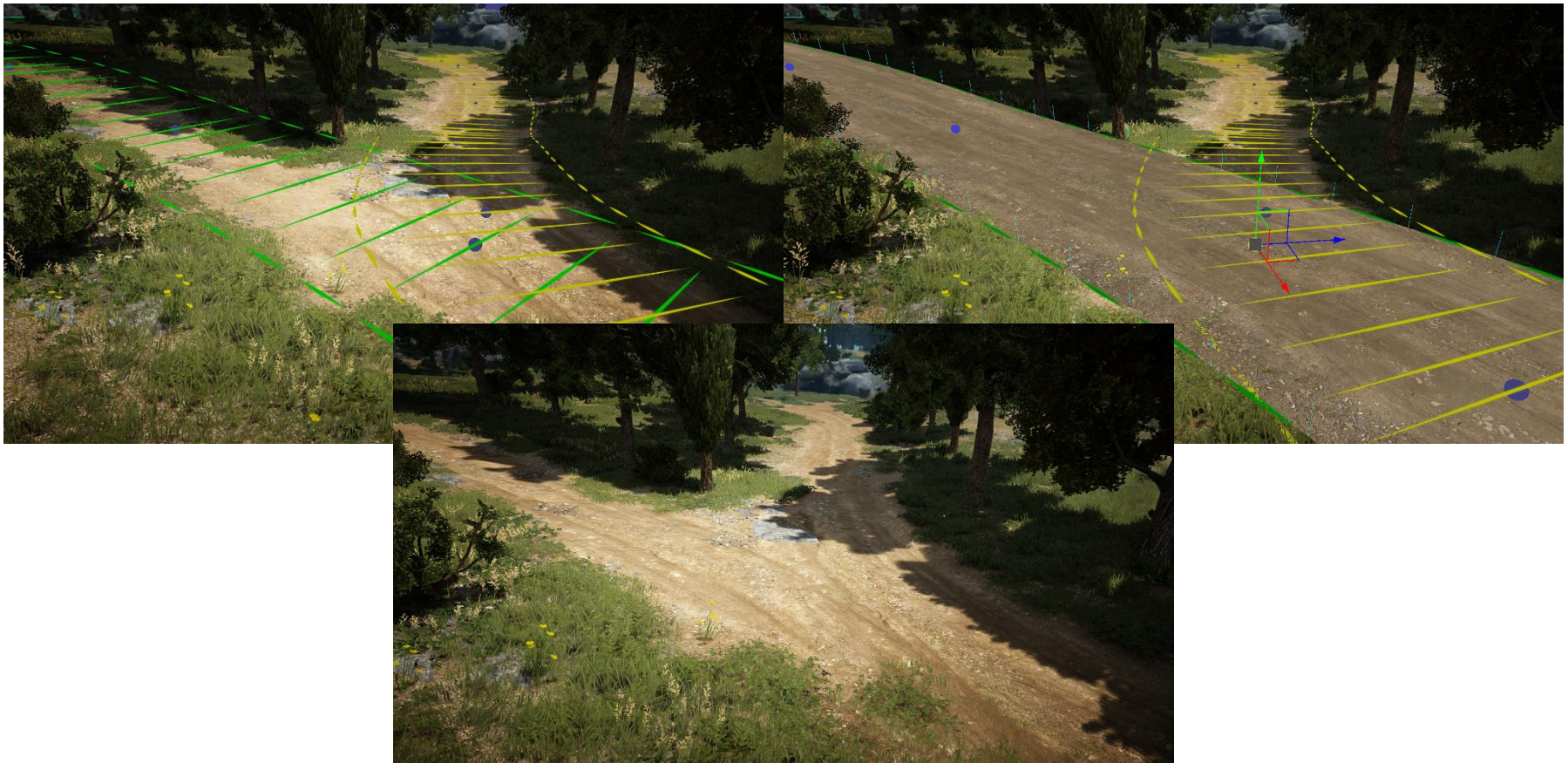
- 데칼 볼륨 생성 before

Decal Volume

- 데칼 볼륨 생성 after

자동화 기능

- LOD, navigation, collision data
  - ✓ 레벨 디자인 된 그대로, **별도 작업 없이** 데이터 생성
- road decal : 지형에 맞게 에디터에서 생성
- ivy vine : 지형, 건물 구조를 따라 자동 생성



3

렌더링 엔진을 활용한 인게임 시스템

igc in 성남

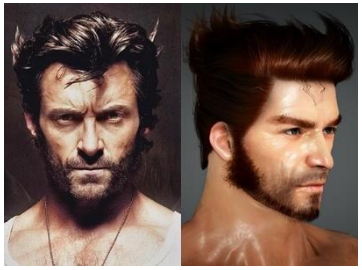
Time of Day (Weather System)

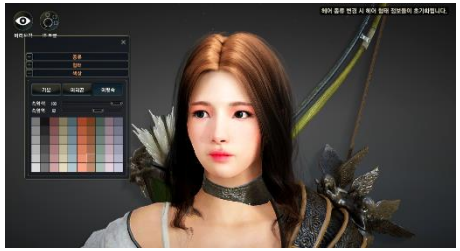
- 24시 설정을 포함하는 20개 테이블(지역, 모드) 시간대 별로 값 조정
  - ✓ scattering parameters
  - ✓ sun / moon의 방향
  - ✓ ambient colors
  - ✓ fog parameters
  - ✓ sea parameters
  - ✓ wind parameters
  - ✓ cloud parameters
  - ✓ color grading parameters
  - ✓ rain, snow parameters

검은사막 Black Desert Online Timelapse

Character Customizing

- 얼굴 형태, 바디 : bone transform
- 문신 : texture uv transform
- 헤어 컬링 : vertex transform
- 대기 상태 표정 : vertex morphing

검은사막 커스터마이징

Event System

- 레벨 오브젝트를 최대한 이용한 이벤트 시스템 개발
- 다양한 환경 변화로 기분 전환과 재미 추구
- 이벤트 시간에 맞춰 오브젝트, 트리, 이펙트, 조명 추가 또는 변경
- 시간이나 날씨도 변경
- 불꽃 놀이, 오로라, 달 등 환경 변화 추가

검은사막 이벤트

Event System

Event System

Event System

Event System

Event System

Event System

Event System

Camera Mode (스샷 모드)

- 캐릭터 시선, 마우스 커서를 따라
- 슬로우 모션
- Depth of Field(DOF) 강도 / 포커스 조절
- Field of View(FOV) 조절
- 포토 필터 : Look-Up Table (LUT)를 이용한 Color Grading
- Post process Effects : 비네팅, 노이즈, 플레어

FOV조절

포토 필터 조절

DOF 조절

4

검은사막 리마스터링

igc in 성남

리마스터링 목표

- 좀 더 사실적이고, 아름다운 화면
- 기존 리소스 수정 없이!
- 기존 옵션은 렌더링 품질, 성능 유지
- 울트라 옵션으로 추가

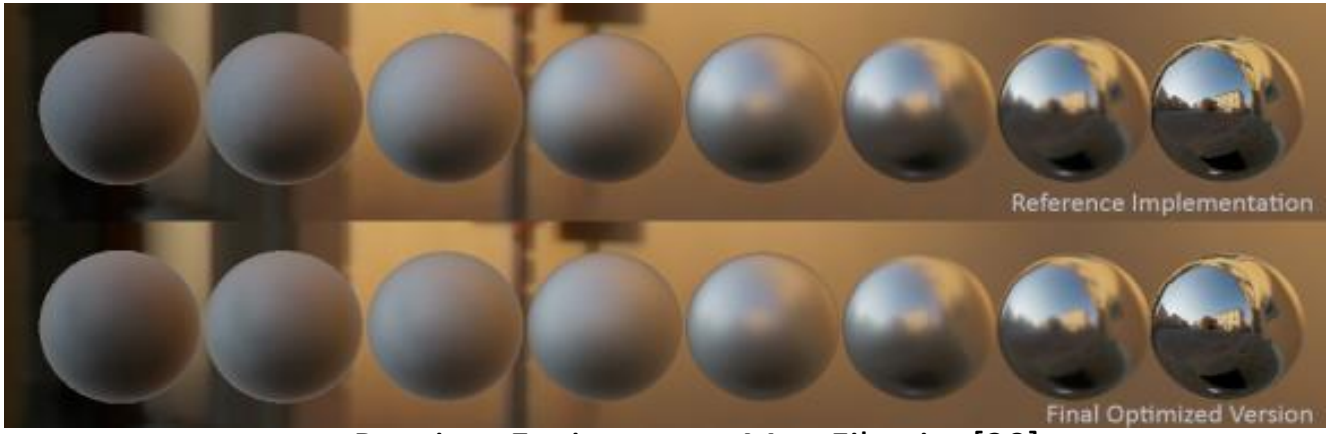
검은사막 리마스터링

리마스터링 영상

무엇을 리마스터링

- **HDR개선**
  - ✓ Float16 RT
  - ✓ Tone mapping, Glare (YEBIS)

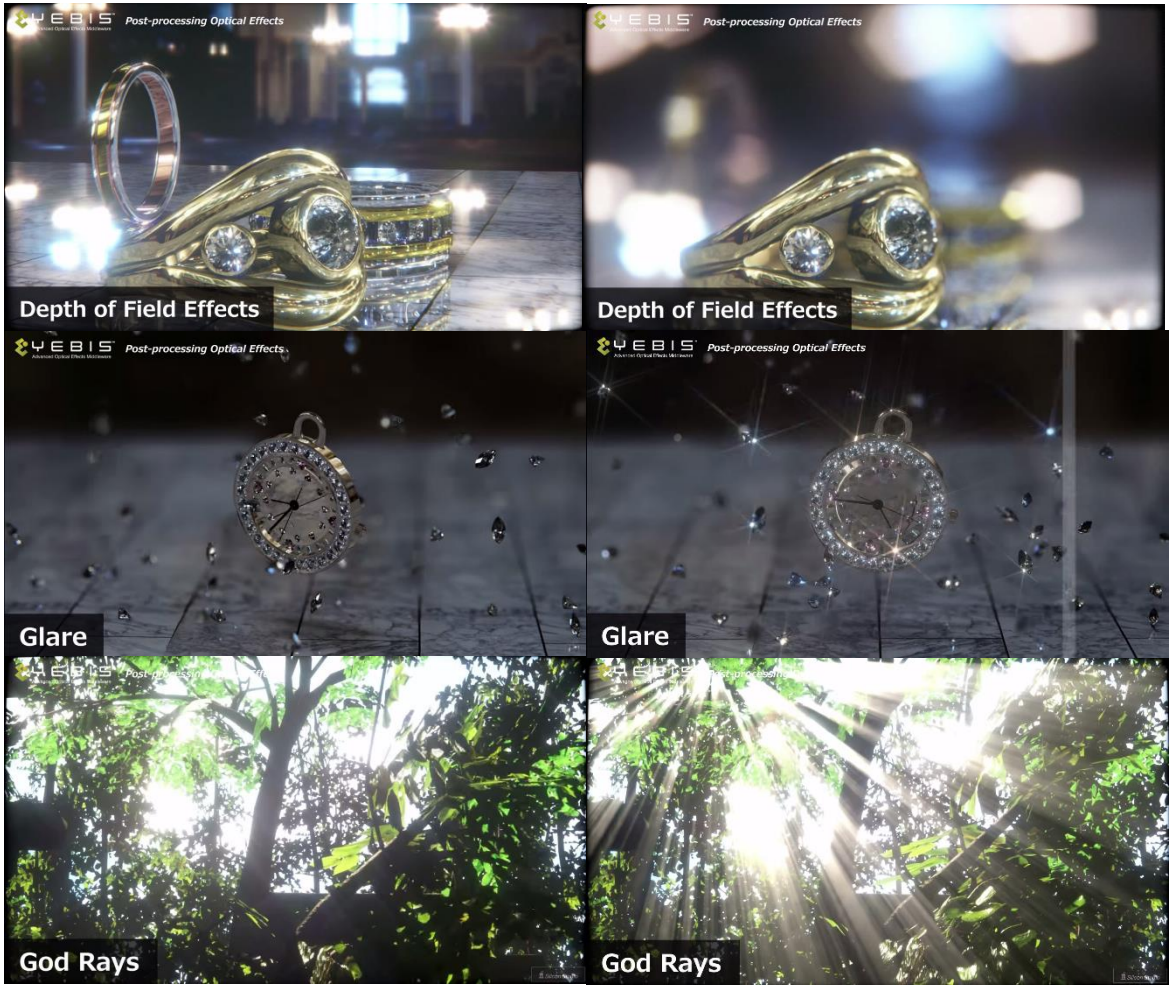
- **PBR 개선**
  - ✓ HDR Environment Map Filtering
    - pre-computed -> runtime
  - ✓ Fresnel, Metallic 개선



Runtime Environment Map Filtering[20]

무엇을 리마스터링

- **Post-process(Motion Blur, Depth of Field, Glare Effects 등)**
  - ✓ YEBIS(Post Processing Effects Middleware)



무엇을 리마스터링

- **SH Lighting** -> Full-size Lighting
- **AO**
  - ✓ HBAO
- **AA**
  - ✓ Temporal AA
- **SSR**
  - ✓ Stochastic Screen Space Reflections + Screen Space Ray Tracing
- **Scattering**
  - ✓ Outdoor Light Scattering by Egor Yusov
- **Volumetric Cloud**
- **Indoor GI**
  - ✓ Voxel Cone Tracing

References

[1] Ryan Geiss, Generating Complex Procedural Terrains Using the GPU
https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch01.html
[2] William E. Lorensen, Harvey E. Cline: Marching Cubes: A high resolution 3D surface construction algorithm. In: Computer Graphics, Vol. 21, Nr. 4, July 1987
http://ab.cba.mit.edu/classes/S62.12/docs/Lorensen_marching_cubes.pdf
[3] Heightmap, Voxel, Polygon (geometry) terrains
https://gamedev.stackexchange.com/questions/15573/heightmap-voxel-polygon-geometry-terrains
[4] Hargreaves, Shawn, and Mark Harris. 2004. "6800 Leagues Under the Sea: Deferred Shading."
http://download.nvidia.com/developer/presentations/2004/6800_Leagues/6800_Leagues_Deferred_Shading.pdf
[5] About YUV Video
https://msdn.microsoft.com/ko-kr/library/windows/desktop/bb530104(v=vs.85).aspx
[6] Lauris Kaplinski, Encoding normals for Gbuffer
http://khayyam.kaplinski.com/2011/07/encoding-normals-for-gbuffer.html
[7] Naty Hoffman, Crafting Physically Motivated Shading Models for Game Development
http://renderwonk.com/publications/s2010-shading-course/hoffman/s2010_physically_based_shading_hoffman_b.pdf
[8] Masaki Kawase, Practical Implementation of High Dynamic Range Rendering
http://www.gdcvault.com/play/1015174/Practical-Implementation-of-High-Dynamic

igc in 성남

References

[9] Naty Hoffman, Background: Physics and Math of Shading
http://blog.selfshadow.com/publications/s2014-shading-course/hoffman/s2014_pbs_physics_math_slides.pdf
[10] Brian Karis, Real Shading in Unreal Engine 4
http://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_slides.pdf
[11] Sébastien Lagarde & Charles de Rousiers, Moving Frostbite to Physically Based Rendering 3.0
https://seblagarde.files.wordpress.com/2015/07/course_notes_moving_frostbite_to_pbr_v32.pdf
[12] hapzunglam, Oren Nayar reflection
https://hapzunglam.wordpress.com/2017/03/13/oren-nayar-reflection/
[13] The Comprehensive PBR Guide by Allegorithmic - vol. 1
https://www.allegorithmic.com/system/files/software/download/build/PBR_Guide_Vol.1.pdf
[14] Schlick 1994, "An Inexpensive BRDF Model for Physically-Based Rendering"
[15] Walter et al. 2007, “Microfacet models for refraction through rough surfaces"
[16] Cook and Torrance 1982, "A Reflectance Model for Computer Graphics"
[17] Eric Heitz, Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs
http://blog.selfshadow.com/publications/s2014-shading-course/heitz/s2014_pbs_masking_shadowing_slides.pdf
[18] Peter-Pike Sloan, Stupid Spherical Harmonics (SH) Tricks
http://www.ppsloan.org/publications/StupidSH36.pdf
[19] ROBERT G. KEYS, Cubic Convolution Interpolation for Digital Image Processing
http://www.ncorr.com/download/publications/keysbicubic.pdf
[20] Padraic Hennessy, Implementation Notes: Runtime Environment Map Filtering for Image Based Lighting
https://placeholderart.wordpress.com/2015/07/28/implementation-notes-runtime-environment-map-filtering-for-image-based-lighting