Render the Possibilities

SIGGRAPH2016

THE **43RD** INTERNATIONAL
CONFERENCE AND **EXHIBITION** ON

Computer Graphics & Interactive Techniques

24-28 JULY

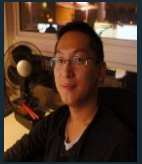
ANAHEIM, CALIFORNIA

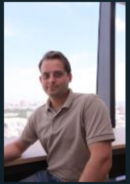
Render the Possibilities
SIGGRAPH 2016
THE 43RD INTERNATIONAL CONFERENCE AND EXHIBITION ON
Computer Graphics & Interactive Techniques

Rendering Techniques of Final Fantasy XV

Masayoshi MIYAMOTO
Remi DRIANCOURT
Square Enix Co., Ltd.

Authors

Sharif Elcott	Advanced Technology Division
Kay Chang	Advanced Technology Division
Masayoshi Miyamoto *	Business Division 2
Napaporn Metaaphanon	Advanced Technology Division
Remi Driancourt *	Advanced Technology Division

* presenters

Other SESSIONS

ANGRY EFFECTS SALAD

Visual Effects of Final Fantasy XV: Concept, Environment, and Implementation

Monday, 25 July, 2-3:30 pm

BUILDING CHARACTER

Character Workflow of Final Fantasy XV

Tuesday, 26 July, 2-3:30 pm

BRAIN & BRAWN

Final Fantasy XV: Pulse and Traction of Characters

Tuesday, 26 July, 3:45-5:15 pm

PLAYING GOD

Environment Workflow of Final Fantasy XV

Wednesday, 27 July, 3:45-5:15 pm

ELECTRONIC THEATER

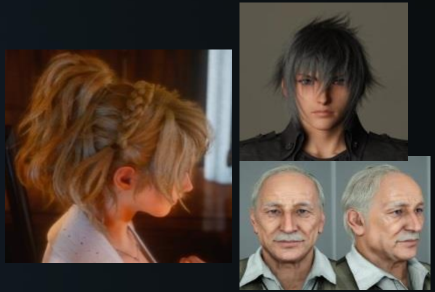
The Universe of Final Fantasy XV

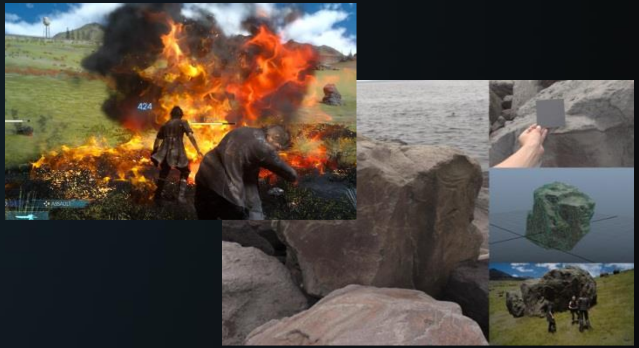
Mon, 25 July, 6-8 pm / Wed, 27 July, 8-10 pm

REAL-TIME LIVE

Real-Time Technologies of FINAL FANTASY XV Battles

Tuesday, 26 July, 5:30-7:15 pm





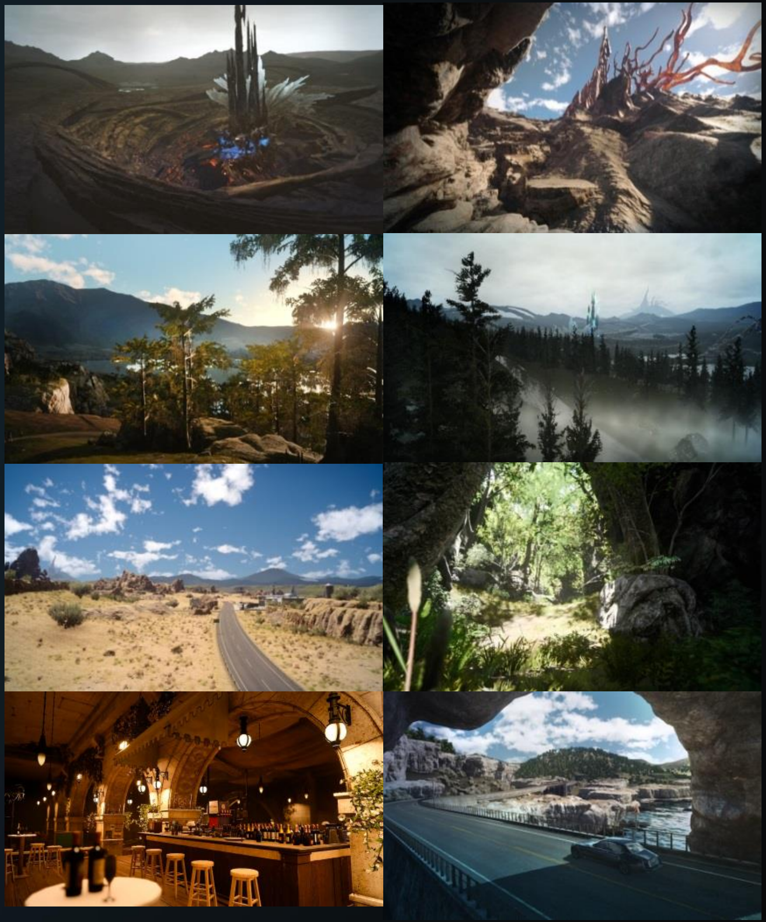
Final Fantasy XV

- Action role-playing game
- PlayStation4, XBoxOne
- Release date: Sept 30, 2016

- Demos
  - Episode Duscae
  - Platinum Demo

Final Fantasy XV

- The most “open-world” FF
  - Indoor & outdoor
  - Day/night cycles
  - Dynamic weather
  - Sky and weather are a big part of storytelling



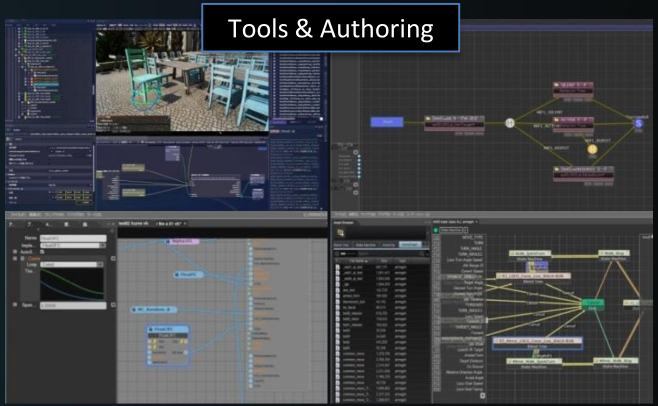

FINAL FANTASY XV
World of Wonder: ENV Footage

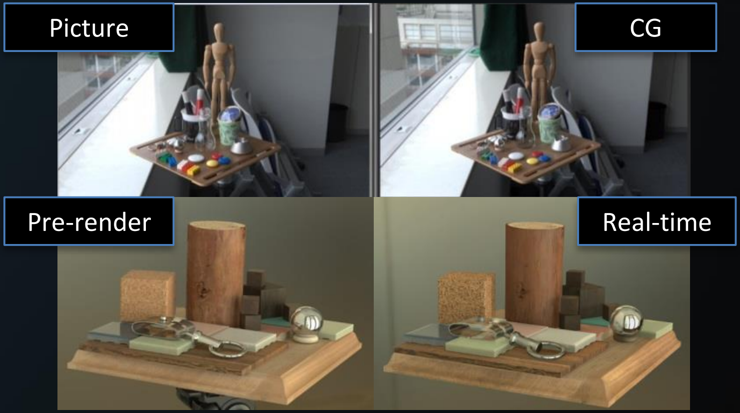
Agenda

- Basic Rendering
- Global Illumination
- Sky
- Weather

Basic features

- Modern AAA-class engine
- The usual suspects
  - Physically-Based Shading
  - Linear workflow
  - Deferred & forward
  - Tile-based light culling
  - IES lights
  - Cascaded Shadow maps
  - Temporal Antialiasing
  - Use of Async Compute
  - Node based authoring
  - etc.





Shading

- Physically-based BRDF model
  - Torrance-Sparrow BRDF
    - Normal distribution function = Trowbridge-Reitz (GGX)
    - Masking function = Schlick-Smith
    - Fresnel term = Schlick
  - Roughness/metallic control
  - Lambertian diffuse

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

$$D_{GGX}(\mathbf{m}) = \frac{\alpha^2}{\pi((\mathbf{n} \cdot \mathbf{m})^2(\alpha^2 - 1) + 1)^2}$$

Shading

Deferred
- Basic BRDF material

\+

Forward
- Transparent
- Special materials

Tricks for special materials

- Eyes: D=diffuse, F=second specular
- Car paint: D=diffuse, F=extra layers (flake/clear coat)
- Skin: D=diffuse+spec(2 RTs), blur diffuse, F=combine
- Hair: D=depth/normal, F=all shading

etc.

G-buffer

G-buffer layout for basic BRDF material

- Used to identify BRDF etc.
- flags for specific processes

	R	G	B	A	
F32_S8	depth				stencil
RT0	compressed normal			roughness	
RT1	albedo			occlusion	
RT2	metallic	-	-	specular	

Used to support special materials (backscatter color, hue shift, etc.)

Agenda

- Basic Rendering
- Global Illumination
  - Indirect Diffuse
  - Specular Reflection
- Sky
- Weather

Lighting: Requirements

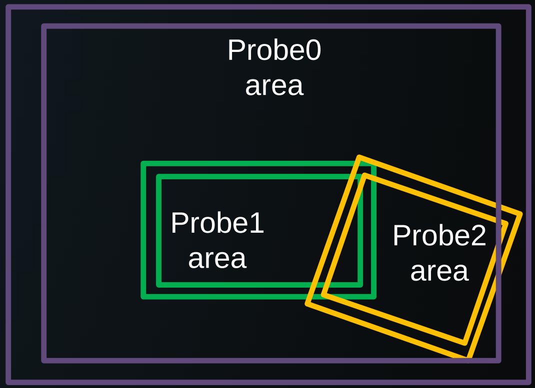
- Seamless indoor/outdoor transitions
- Moving vehicles (e.g. train cars)
- Time of Day
- Dynamic Weather

- Cannot rely only on static baked lighting data
- Data storage requirements

➡ Hybrid GI strategy based on both dynamic/static data.

Indirect Diffuse: Local probe

- We use grids of local light probes
  - Can be placed to fit navigation meshes or heightmaps automatically
- Organized into hierarchies of grids
  - overlapping light probe grids
  - $w_1 C_1 + \cdots + w_N C_N + (1 - (w_1 + \cdots + w_N))K$
    - w_i:weights, C_i: diffuse by probe, K: diffuse by sky
  - Controls of fade-out regions
  - Controls of blending priorities



Indirect Diffuse: Local probe

A given probe grid can have either:

- Precomputed Radiance Transfer (PRT)
  - Sky occlusion
  - fully outdoor scenarios
- Irradiance Volumes (IV)
  - Diffuse lighting from static local lights
  - fully indoor scenarios
- Both
  - outdoor towns, rooms with windows, etc...

Indirect Diffuse: Local probe

- PRT transfer matrices calculated by in-house path tracer
  - Matrix of SH coefficients (order 3 SH)
- Both IV and PRT data stored together
  - IV (order 3 SH) is just an additional row to the PRT transfer matrix

At runtime, we light the probes:

(sky SH) * (PRT transfer matrix) + (irradiance SH)

Run time data

Baked data

For each 9 coefficients of SH
For each RGB channel.

Render the Possibilities
SIGGRAPH2016

SQUARE ENIX

Indirect Diffuse: Moving probe

- Handling moving environments
  - Trains and airships have probes inside.
  - The environment itself can move and rotate relative to the outside

Probes can be attached to train wagons

Solution:

Bake probes with local environment
Rotate SH data at run time based on relative orientation to the sky.

Indirect Diffuse: Local occlusion

- Screenspace Ambient Occlusion
  - Custom algorithm created in collaboration with the LABS group [Michels et al. 2015]
  - Half-res AO/blur with upsampling
  - 8 samples
  - Use temporal reprojection
- Analytical AO
  - "AO spheres" attached to foliage & heroes
  - Apply analytical AO in tiled fashion
    - less than 1ms

Indirect Diffuse: Local occlusion

- Screenspace Ambient Occlusion
  - Custom algorithm created in collaboration with the LABS group [Michels et al. 2015]
  - Half-res AO/blur with upsampling
  - 8 samples
  - Use temporal reprojection
- Analytical AO
  - "AO spheres" attached to foliage & heroes
  - Apply analytical AO in tiled fashion
    - less than 1ms

Indirect Diffuse: Dynamic light bounce ?

- We disabled RSM(Reflective Shadow Map)-based dynamic GI.
  - Light Propagation Volumes [Kaplanyan 2010]
  - Virtual Spherical Gaussian Lights [Tokuyoshi 2015]

- Completely dynamic, but RSMs were expensive for our game.
  - Lots of high-poly scenes in our game.
- Instead, we use computational resources for rendering dynamic natural environment like sky/clouds.

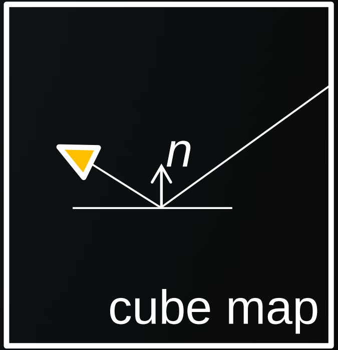
Agenda

- Basic Rendering
- Global Illumination
  - Indirect Diffuse
  - Specular Reflection
- Sky
- Weather

Specular Reflection

Tiered system

- Global cubemap (sky box)
  - Updated by time of day and weather
  - Filtering is spread over multiple frames
- Local cubemaps with parallax correction
  - We want to support time-of-day change
- Screenspace reflection [Wronski 2014, Valient 2014]
  - Classic ray march
  - Roughness-based bilateral blur (half-res)& upsample



Specular Reflection

- Problems with classic local cubemaps
  - Static. How to handle time-of-day / weather changes?
  - Baking probes at runtime is too expensive.
- Typical workaround:
  - 1. Re-light cubemap at runtime using a mini G-buffer [McAuley 2015]
    - Still expensive
  - 2. Blend between probes baked in different time/weathers
    - Blending artifacts

Specular Reflection

Our solution

- Split into 3 components:
  - 1. Sky pixel
  - 2. Pixel not affected by time of day (local lights / emissive)
  - 3. Pixel affected by time of day (e.g. sun and sky)
- Fast to evaluate
- Less memory footprint.

Specular Reflection

- 1st component: Sky Mask
  - At bake time
    - Generate mask that identifies “sky” pixels
  - At runtime
    - shaders fall back to the dynamic skybox based on this mask
    - reflection vectors that hit the sky see moving clouds!

Specular Reflection

- 2nd component: Baked static lighting
  - At bake time
    - Turn off : sun, skylight, fog, atmospheric scattering, etc..
    - Do a cubemap capture & filter
  - At runtime
    - Use the map as-is, with a roughness-based lookup
    - same as in [Valient 2014]

Specular Reflection

- 3rd component = Sun and Sky
  - At bake time
    - Keep all lights ON and render
    - Substract previous “local lighting” to get a cubemap of the scene as lit by only the sun and sky
    - Sky color (constant term of sky SH) divided out before saved
  - At runtime
    - Sky color for the current frame is multiplied back in

Specular Reflection

- Summary

dynamic data

Constant term of the SH-projected sky box

Sky box color

$$\text{Reflection} = (S_n D_r + L)(1 - M) + K_r M$$

static data

Normalized baked sky data

Baked local light data

Baked sky mask

Specular Reflection

- Data storage
  - If we implement naïvely, we would need 7 channels
    - S_n : sun and sky lighting, RGB (HDR)
    - L : baked local light data, RGB (HDR)
    - M : sky mask, float {0, 1}
  - Of course, we will compress

Specular Reflection

Key idea

- Assume that S_n and L have roughly similar color.

$$\text{Reflection} = \boxed{(S_n D_r + L)}(1 - M) + K_r M$$

$$\boxed{S_n(D_r + R)} \quad \text{with } R = L/S_n$$

We approximate the ratio R with a single channel.

Note that S_n can be zero indoor, and L can be zero outdoor.

Specular Reflection

Solution

1. Pick S or L as the key color

$$
\begin{aligned}
\text{if}(S = 0 \text{ and } L = 0) \qquad & key = 0, 0, 0 \\
& R = 0 \\
\text{else if}(S \geq L) \qquad & key = S \\
& R = lum(L)/lum(S) \\
\text{else} \qquad & key = L \\
& R = lum(S)/lum(L)
\end{aligned}
$$

Specular Reflection

Solution

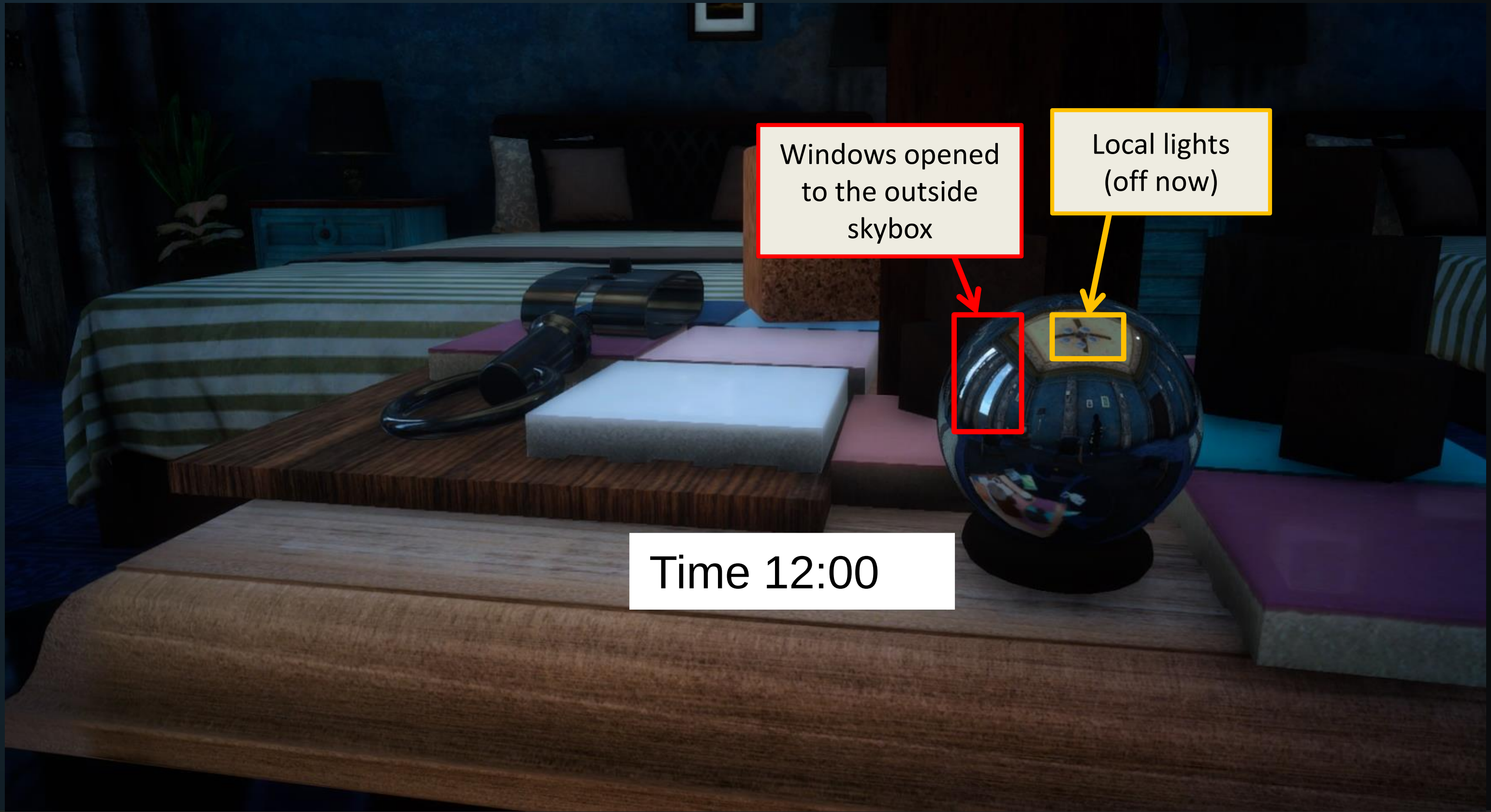
1. Pick S or L as the key color
2. Disambiguate at runtime using [0-2] range

$$
\begin{aligned}
\text{if}(S = 0 \text{ and } L = 0) \quad & key = 0,0,0 \\
& R' = 0 \\
\text{else if}(S \geq L) \quad & key = S \\
& R' = lum(L)/lum(S) \\
\text{else} \quad & key = L \\
& R' = 2 - lum(S)/lum(L)
\end{aligned}
$$

If $R' \in [0,1]$ then the key is S and $R = R'$.

If $R' \in [1,2]$ then the key is L and $R = 2 - R'$.

Specular Reflection: results



Specular Reflection: results

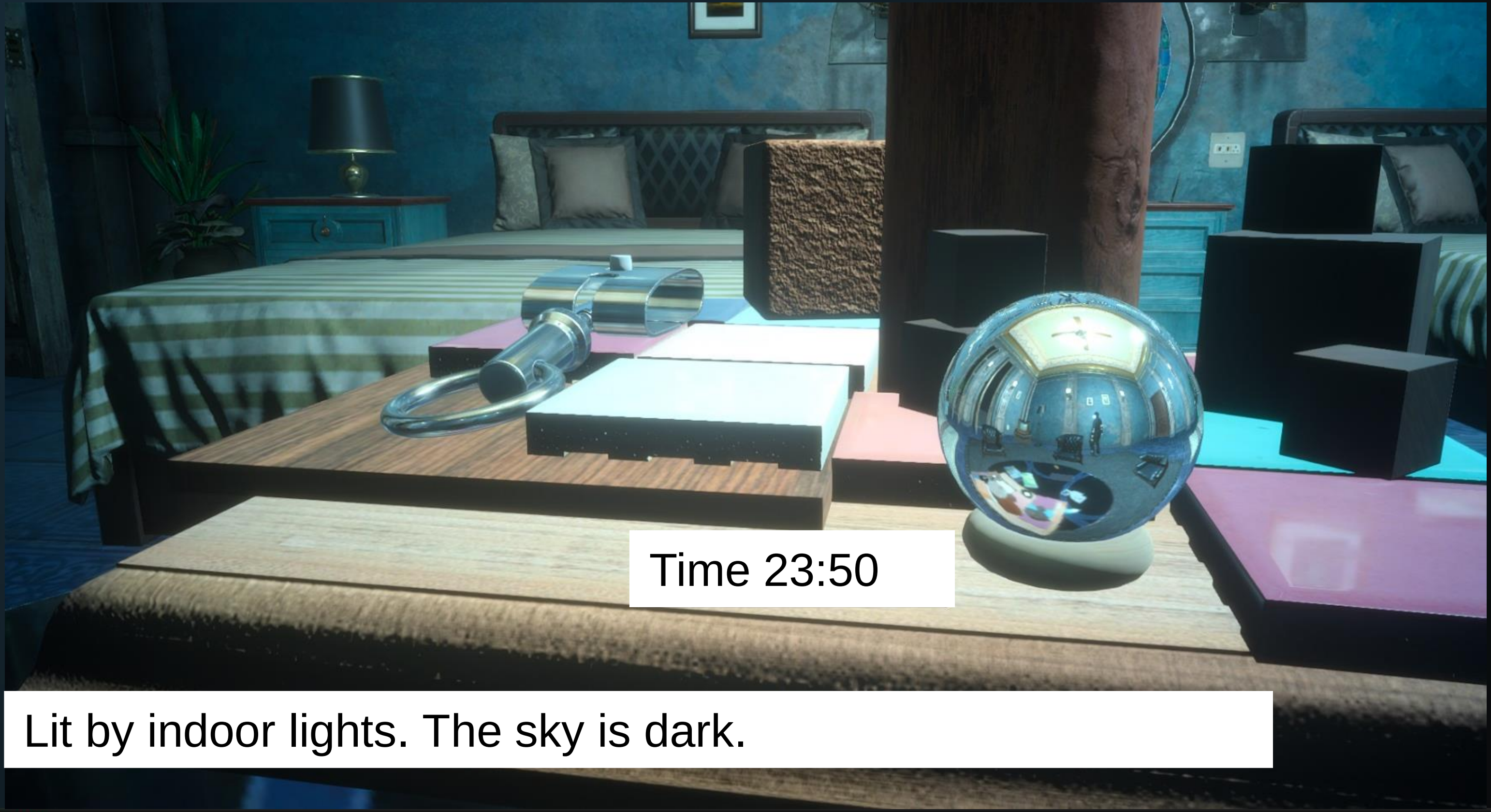
Specular Reflection: results

Specular Reflection: results

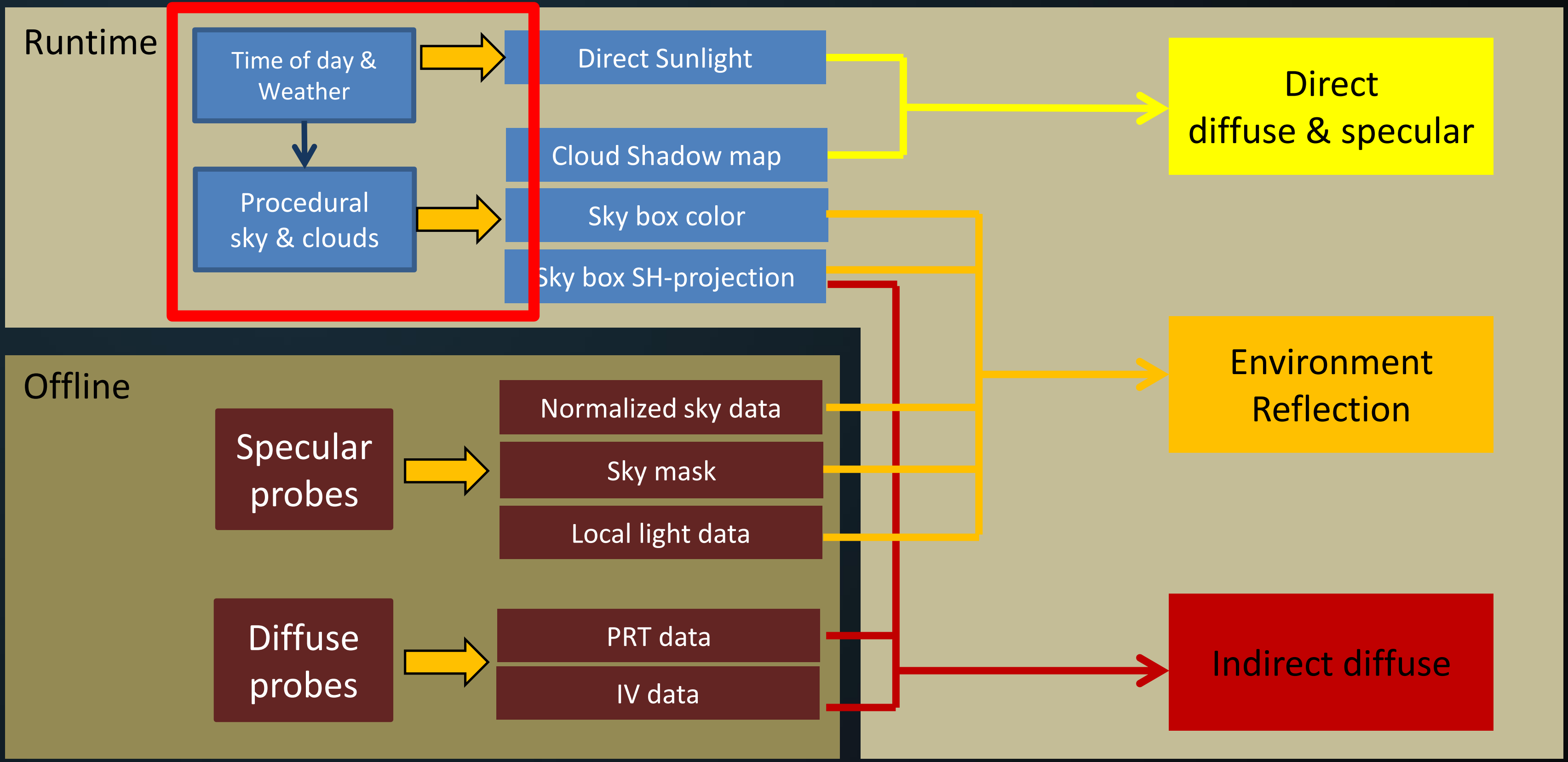
Time 19:20

Sunset. The scene is still lit by sky. Indoor lights switched on.

Specular Reflection: results



Summary

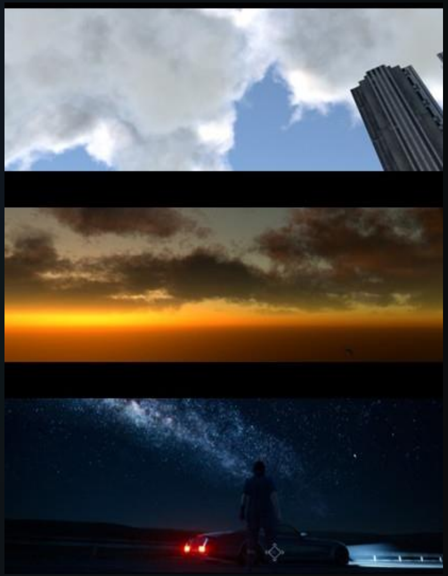

Runtime
Time of day & Weather
Procedural sky & clouds
Direct Sunlight
Cloud Shadow map
Sky box color
Sky box SH-projection
Offline
Specular probes
Diffuse probes
Normalized sky data
Sky mask
Local light data
PRT data
IV data
Direct diffuse & specular
Environment Reflection
Indirect diffuse

Agenda

- Basic Rendering
- Global Illumination
- Sky
  - Sky
  - Atmospheric Scattering
  - Clouds
  - Sky Cubemap
- Weather

Sky: Requirements

- Dramatic changes in atmosphere
- Linked with lighting and weather
- Quick but smooth transitions
  - from a cloudy afternoon to a rainy evening
  - from a clear starry sky to a red dawn
- Artist-Directable



Approach

- Generate the whole sky procedurally
  - Sun
  - Stars
  - Moon
  - Clouds
  - Atmospheric scattering

Sky Rendering

- Layers of sky:
  - Celestial objects
    - Milky way: on a cylindrical surface
    - Small stars: repeated textures
    - Large stars: billboards, instanced
    - Moon
    - Sun
  - Clouds + Atmospheric scattering (sky)
  - Atmospheric scattering (aerial perspective, fog to objects)

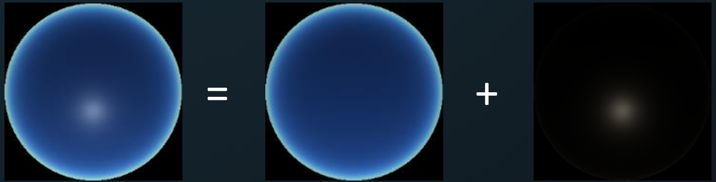
Atmospheric Scattering

- Standard models in games:
  - Single scattering model [Hoffman, Preetham, 2002]
    - + Fog can be rendered with the sky
    - - No twilight. Completely dark after sunset.
    - - Unintuitive artistic controls.
  - Precomputation/Analytical model [Bruneton, 2008], [Preetham, 1999], [Hosek, 2012]
    - Better artistic controls.
    - Didn’t match our reference photo well enough.

      -- we chose this model, but made static data on our own.

Atmospheric Scattering (Sky)

- Strategy: precomputed approach
  - Combine of LUTs (Lookup tables) and Rayleigh/Mie scattering function
  - Sky $= LutR(\theta,\gamma) * phaseR(\mu) + LutM(\theta,\gamma) * phaseM(\mu, g)$
    - θ: sun-zenith angle, γ: view-zenith angle, μ: angle from the sun
    - $phaseR = 1 + \cos^2 \mu$, (constants omitted for simplicity)
    - $phaseM = \frac{1-g^2}{2+g^2} * phaseR * (1 + g^2 - 2g\cos\mu)^{-1.5}$, g: ”haziness” in [0,1)



sky “Rayleigh part” “Mie part”

Atmospheric Scattering (Sky)

- Generate LUTs offline
  - A least squares fitting
  - Ray-trace sky (inscatter) based on real sky database.

- We dropped: High level view of sky, Earth’ shadow, etc.
  - Instead, we use the simple sky formula/LUTs.
  - but it was enough for our game.

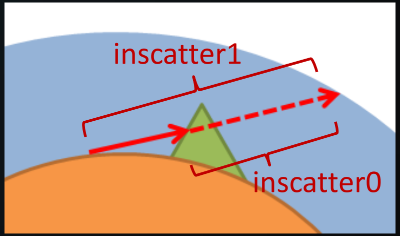
- Special case: Overcast sky
  - It uses different model [ISO 2004]
  - So we ended up mixing two models.

$\frac{1+2\sin\gamma}{3} * L_{zoc}$, L_{zoc}: zenith luminance

Atmospheric Scattering (Aerial perspective)

- Aerial perspective
  - *(inscatter color) + (transmittance) * (object color)*
- Transmittance: Beer's law
  - *(transmittance) = exp(-a*distance)*
- Inscatter color
  - (above pic) Bluish color addition in the distance.
  - In theory, inscatter is diff of sky colors at two points

    inscatter1 – inscatter0
  - We cannot get inscatter0 from our sky LUTs.
    - We made different LUTs.



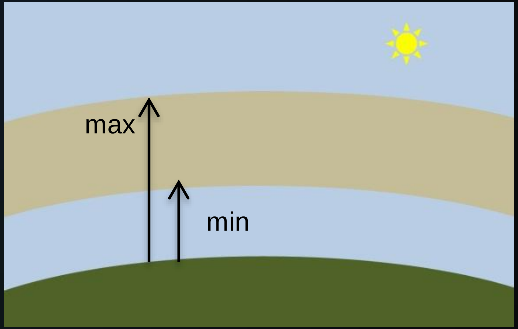
Atmospheric Scattering (Aerial perspective)

- Inscatter for aerial perspective
  - Inscatter only in horizontal direction.
  - Incatter is a combination of:
    - Mid-ground LUTs and a background LUT , for each Rayleigh/Mie component.
    - LUTs are combined with B-spline basis functions.
  - Background LUT is horizon part of sky LUT.
  - Designers can tint mid-ground color.

Clouds

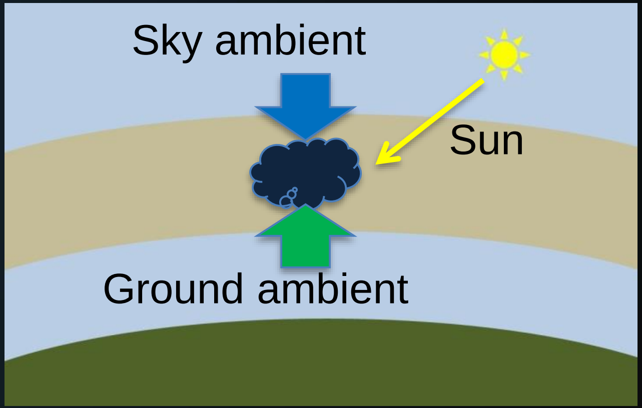
Clouds: Modeling

- Range
  - Above the camera position
  - Within user-defined range (altitude min/max, horizontal radius)
- Density Function
  - Combination of noise with 7 octaves
    - Different amplitude/animation speed for each.
    - Lowest octave -> rough cloud shape.
    - 2 lowest octaves -> rough shape animation
    - Higher octaves -> details.
  - Noise is obtained by sampling a small 3d texture.
  - Lots of parameters exposed to designers to make variation.



Clouds: Lighting

- Light clouds by raymarching
  - Three light sources:
    - Direct light from sun/moon
    - ambient from above (sky)
    - ambient from below (ground).

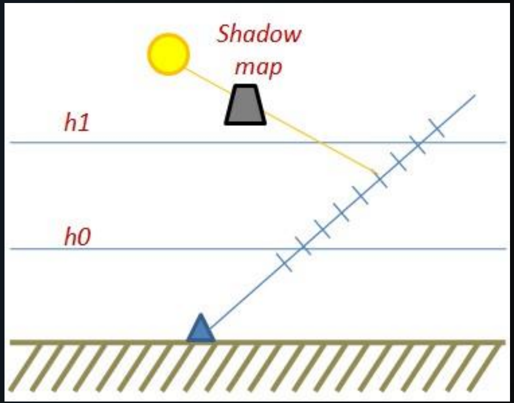


- Cloud opacity
  - is also calculated along with raymarching.
  - in order to blend the clouds with the sky dome.

- 4 results are packed into a RGBA8 texture.

Clouds: Lighting

- Direct light from sun/moon
  - Ray march with single scattering model.
  - We don’t use the 2nd ray march to the sun
  - Instead, we use *extinction transmittance map*
    - how much the light reaches the sample [Gautron, et al. 2011]



- Ambient is analytically computed
  - Integral over hemisphere assuming that the density is constant.

Clouds: Lighting

- Mie Scattering
  - Scattering phase function is factored out from the ray march.
  - Phase function gives directionality to the lighting.
    - Clouds close to the sun are brighter

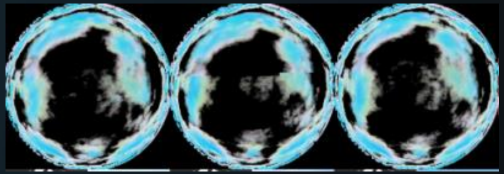
Clouds: Shadow

- ETM (Extinction Transmittance Maps)
  [Gautron, et al. 2011]
  – Clouds' self shadow
  – Shadow on the surface

- Transmittance curve along sun ray.
- A curve is encoded into:
  – 4 values using DCT (Discrete Cosine Transformation)
  – 2 values for start/end point of the curve
- These values are packed into 2 textures.

Clouds: Implementation

- Data storage
  - Raymarching results
    - 1536x1536, RGBA8 texture x3

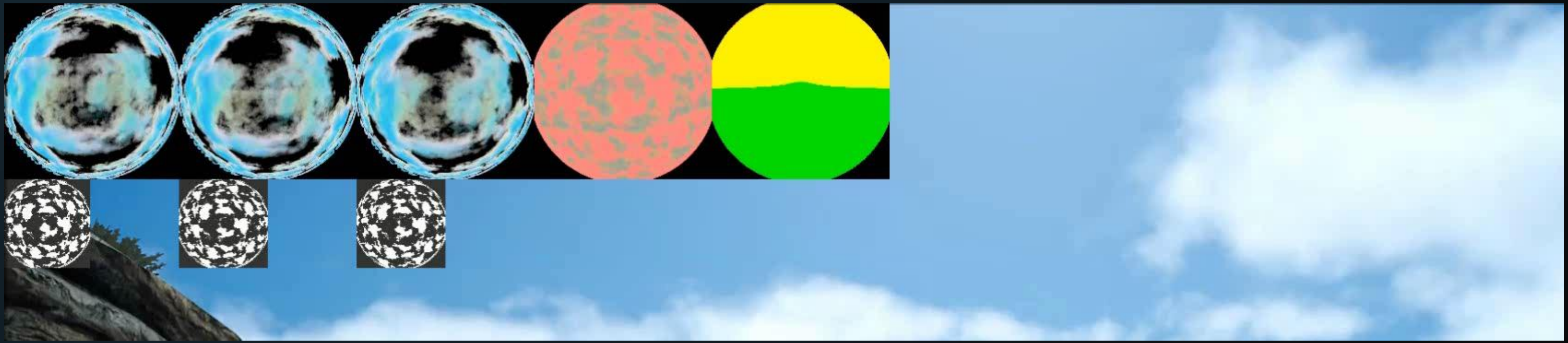


  - ETM
    - 512x512, RGBA8 texture
    - 512x512, RG16 texture

  - Shadow map for ground
    - 512x512, R8 texture x3

Clouds: Implementation

- We amortize the cost across several frames.
  - Raymarching : sky dome is split into 64 slices. 1 slice update/ frame.
  - ETMs for 4 frames.
  - Shadow map for ground.
- Async compute



Clouds: Sky dome

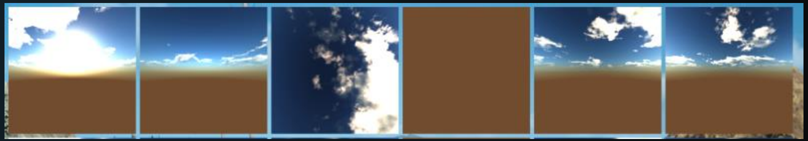
- Project clouds onto sky dome:
  - Cross-fade two cloud raymarch results over time
    - While the other one is being updated.
  - Wind Animation

Clouds: Wind

- Wind animation
  - 7 different animation speed in density function.
    - Choose dominant speed.
  - Assume clouds travel at a given fixed height.
  - Take into account perspective
    - Far clouds travels at a slower pace
    - Aerial perspective
- Animate cloud shadow in the same way

Sky Cubemap

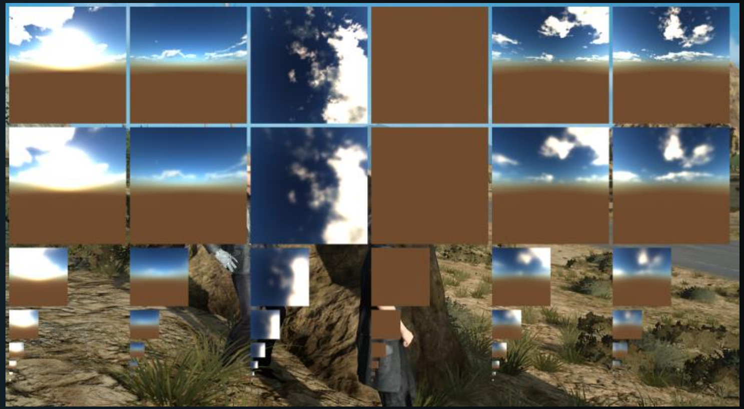
- Dynamic sky cubemap
  - Render only sky & clouds into cubemap
  - Lower hemisphere is user-defined “ground”
    - Lit by sky/sun as diffuse material



- SH Projection of Cubemap
  - For diffuse lighting by skylight.
  - Can be combined with PRT.

Sky Cubemap

- BRDF Filtering of cubemap
  - 1 face/1 mip updated per frame
  - total cycle: 48 frames (6 faces x 8 mips)



- Frame rate stability
  - Smaller mip levels require more filter kernel samples
  - Hence the filtering cost per frame stays roughly constant.
- Problem: fast change of sky (e.g. around sunset)
  - Sky cubemap's intensity is divided by total sky luminance.

Exposure

- Problem
  - Wide range of luminance
    - Sun luminance = Moon luminance * 400,000
    - We use real sky values by default.
  - Floating point precision issue.
  - Brightest pixels can be clamped.

- Solution
  - Exposure value is multiplied to all light sources, sky, etc.
  - Not apply to screen at the post-processing stage.

Results

Results

Results

12:00

Results

Results

Calibration by artists

- Photo shooting
  - Actual cubemap reference
  - 24-hours straight in HDRI

- Lighting for game
  - Calibrated each parameter based on the photo data

IBL Procedural sky

Agenda

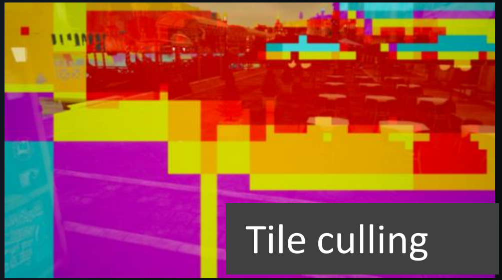
- Basic Rendering
- Global Illumination
- Sky
- Weather
  - Volumetrics
  - Rain
  - Wind
  - Weather System

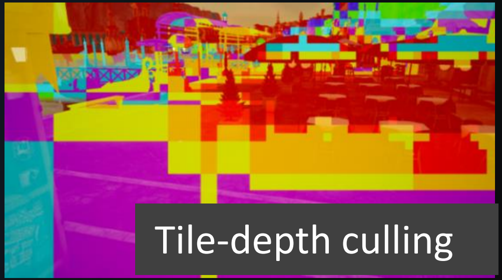
Volumetrics (fog, light shafts)

- 3d grid (160x90x64)
  - System like [Wronski, 2014]
  - Range: normally ~100m
    - Use different system for distant fog
- Lighting (optional per light)
  - Directional light
  - Local lights
  - Light probes
- Sample blurred shadow that makes light shafts
- Noise, wind animation, etc.

Local Lights

- Tile/Depth culling
  - 32 depth slices, logarithmical
  - Low-res depth min/max texture
  - Each tile has a light list, and each cell has min/max of light list.
  - Light probes use tile culling.
- Local shadow map
  - Dynamic resizing of local shadow map.
  - Texture atlas

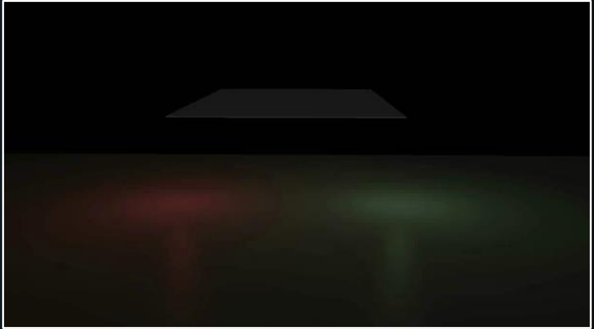




Color shows the number of lights in tile

Rain

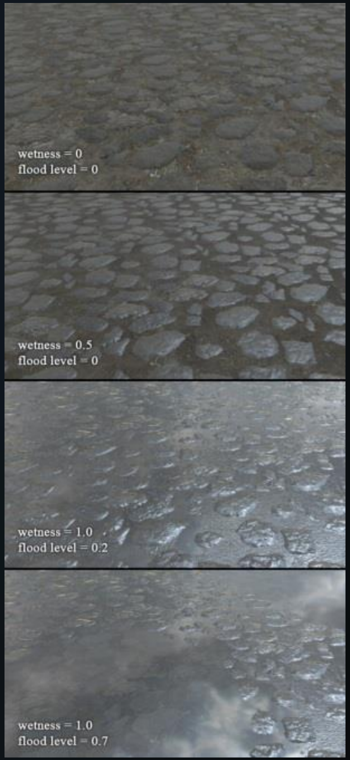
- Rain drops
  - GPU particle system
  - Falling particles centered on camera
  - Depth map collision
    - Render depth map from top view
  - Splash particles emitted from the surface



- Character Interaction

Wet Materials

- Wet shader permutation
  - Almost everything can get wet.
  - Wetness [Lagarde 2012]
    - Increases specular
    - Decreases roughness
    - Darkens diffuse
    - Distortion of normal



Wet Materials

- With in-house shaders, we made:
  - Puddles
  - Ripples
  - Trickling water
  - etc.

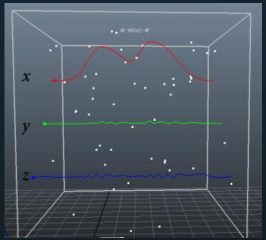
Rain: Problems

- Camera sometimes moves too fast.
  - e.g. Player character (Noctis) can warp
  - Solution: We shift particle positions during simulation when that happens.

- Wetness to dynamic objects
  - Characters/vehicles can get dry instantly when moving under a roof.
    - Because we use depth map from top view.
  - Solution: designers can place “Non wet” box.
    - Wetness decreases smoothly when they moved into a box.

Wind

- Wind system
  - Affects bone-based simulation
    - e.g. cloth, hair, fur
  - Affects vertex-based procedural motion
    - e.g. vegetation, fur
  - Wave functions can be “drawn” by designers



Weather System

- Links “weather” (set of parameters) with game entities
  - Parameters: sky, rain, wind, etc.
    - Affect shading, vfx, animation, etc.
  - Entity: area box, sequence node
- Animates parameters and changes weather.

TIME_OF_DAY	0	1	2	3	4	5	6	7	8	9	10	11	12	1
VOLLIGHT_FARFOG_THICKNESS	2000			2000	0.5								0.5	
VOLLIGHT_FARFOG_ZNEAR	200													
VOLLIGHT_FARFOG_ZFAR	8000													
VOLLIGHT_FARFOG_HEIGHT_DECAY	0.3													
VOLLIGHT_FARFOG_DISTANCE_DECAY	0.02													
VOLLIGHT_FARFOG_DEPTH_DECAY	0.3													
VOLLIGHT_FARFOG_EXTINCTION	0.001				0.001			5					5	
VOLLIGHT_FARFOG_TINT_R	0.5				0.5	0.23							0.23	
VOLLIGHT_FARFOG_TINT_G	0.5				0.5	0.5							0.5	
VOLLIGHT_FARFOG_TINT_B	0.5				0.5	0.82							0.82	
VOLLIGHT_FARFOG_STANDARD_HEIGHT	0													
VOLLIGHT_FARFOG_BOTTOM_HEIGHT	-100													
VOLLIGHT_FARFOG_PLAYER_RELATIVE	0													
RAIN_PARTICLE_EMIT_RATE	0													
RAIN_DEPTH_MAP_RADIUS	100													
RAIN_EMIT_RADIUS	30													
RAIN_EMIT_HEIGHT	50													
RAIN_EMIT_SPEED	40													
RAIN_EMIT_SPEEDRANDOM	10													
RAIN_EMIT_SPREAD	0.003													
RAIN_LIFE_SPAN	3													

a weather file

Special Thanks

Ivan Gavrenkov
Chou Ying-I
Pavel Martishevsky
Christina Haaser
Shawn Wilcoxen
Benedict Yeoh
Yusuke Tokuyoshi

Akira Iwata
Kimitoshi Tsumura
Seiji Nanase
Hiromitsu Sasaki
Takashi Sugata
Tomoharu Oiyama
Takeshi Aramaki
Yusuke Hasuo
Takashi Sekine

References

Global Illumination

- MCAULEY, S. 2015. Rendering the world of Far Cry 4. In Game Developers Conference 2015.
- SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, lowfrequency lighting environments. ACM Trans. Graph. 21, 3 (July), 527–536.
- KAPLANYAN, A., AND DACHSBACHER, C. 2010. Cascaded Light Propagation Volumes for Real-Time Indirect Illumination. In Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, ACM, 99–107.
- TOKUYOSHI, Y. 2015 Fast Indirect Illumination Using Two Virtual Spherical Gaussian Lights. In ACM SIGGRAPH ASIA 2015 Posters, pp.12:1-12:1 (2015)
- MICHELS, A. K. Et al. 2015. Labs R&D: Rendering techniques in Rise of the Tomb Raider. In ACM SIGGRAPH 2015 Talks, ACM, New York, NY, USA, SIGGRAPH '15
- VALIENT, M. 2014. Reflections and volumetrics of Killzone Shadow Fall. In Advances in Real-Time Rendering in Games, ACM SIGGRAPH 2014 Courses

Atmospheric Scattering, Cloud, Fog, Rain

- BRUNETON, E., AND NEYRET, F. 2008. Precomputed atmospheric scattering. Computer Graphics Forum 27, 4, 1079–1086.
- HOFFMAN, N., AND PREETHAM, A. J. 2002. Rendering outdoor light scattering in real time. In Game Developers Conference 2002.
- PREETHAM, A. J., SHIRLEY, P., AND SMITS, B. 1999. A practical analytic model for daylight. In Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, ACM SIGGRAPH 1999, 91–100.
- GAUTRON, P., DELALANDRE, C., AND MARVIE, J.-E. 2011. Extinction transmittance maps. In SIGGRAPH Asia 2011 Sketches.
- SCHNEIDER, A. 2015. The real-time volumetric cloudscapes of horizon: Zero dawn. In Advances in Real-Time Rendering in Games, ACM SIGGRAPH 2015 Courses, SIGGRAPH '15.
- L HOSEK, A WILKIE, J. L. 2012. An analytic model for full spectral sky-dome radiance. ACM Transactions on Graphics (TOG) 31 (4), 95.
- WRONSKI, B. 2014. Assassin's Creed 4: Road to next-gen graphics. In Game Developers Conference 2014.
- LAGARDE, S., 2012. Water drop series. https://seblagarde.wordpress.com/2012/12/10/observe-rainyworld/.

Thank You

Questions ?

Extra slides for Q&A

Specular Reflection

Another solution: lerp between indoor/outdoor case

$$\boxed{(S_n D_r + L}\,)(1 - M) + K_r M$$

$$S_n D_r + L \approx (S_n + L) * \frac{lum(S_n) * D_r + lum(L).\mathrm{xxx}}{lum(S_n + L)}$$

$$= keycolor \; * (ratio \; * D_r + 1 \; - ratio)$$

where $keycolor = S_n + L$

$ratio = lum(S_n) \, / \, lum(S_n + L)$

Rendering pipeline

- Use of Async pipes