Game Graphics Techniques

PART I

Georgios Papaioannou - 2020

DEFERRED APPROACHES

Deferred Rendering - The Principle

- Deferred shading defers (postpones) most of the heavy rendering (like lighting) to a later stage
- Deferred shading consists of two passes:
  - The geometry pass renders the scene once and retrieves all kinds of geometrical information from the objects that we store in a collection of textures called the G-buffer
  - In the lighting pass, we render a screen-filling quad and calculate the scene's lighting for each fragment using the geometrical information stored in the G-buffer

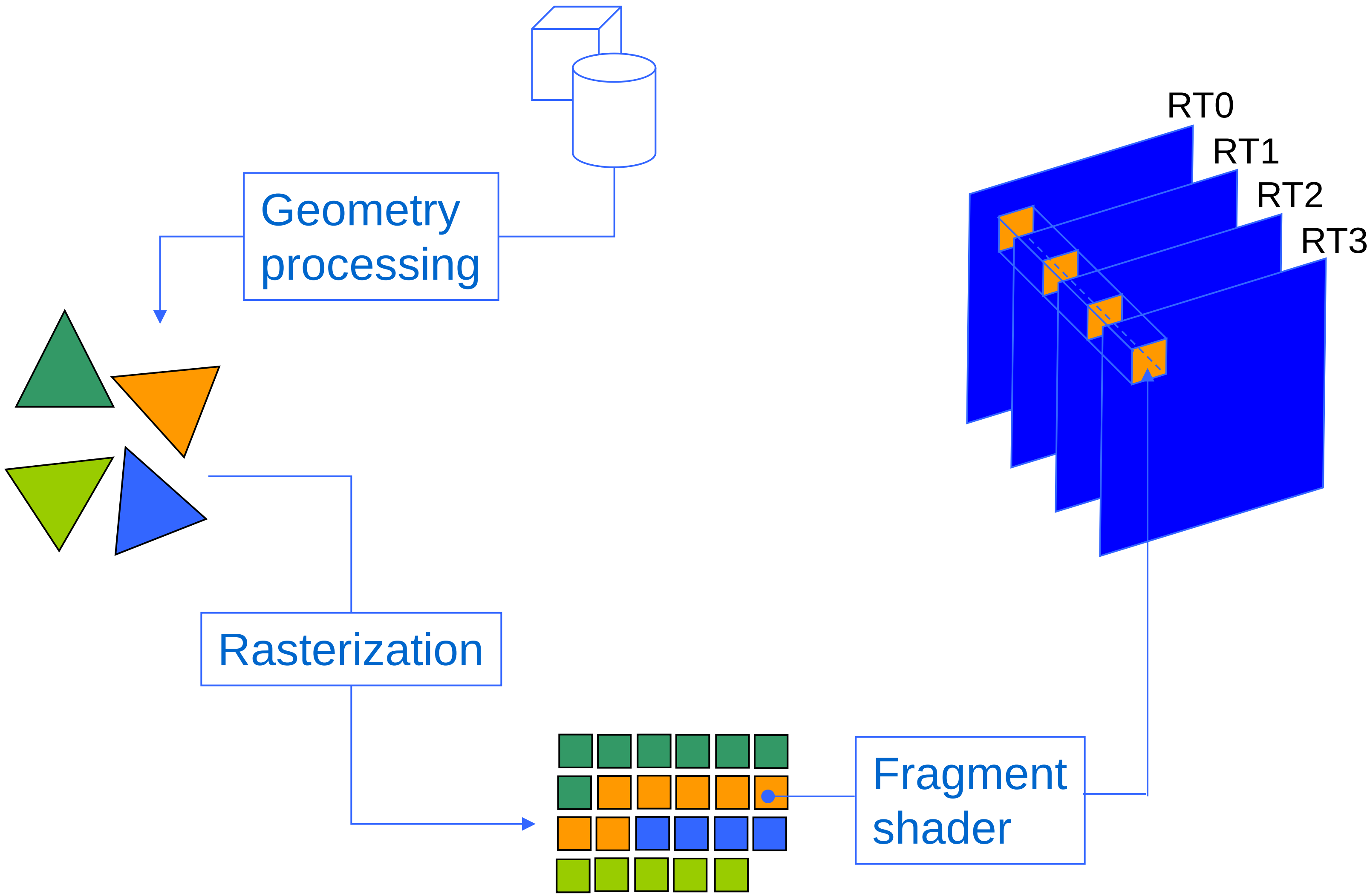
Multiple Render Targets (1)

- It is often useful to be able to write many fragment operation results to **multiple internal buffers, without re-rendering the geometry**
- Examples:
  - Cube map generation (6 buffers, 6 viewing transformations – also requires retargeting by a geometry shader)
  - Deferred rendering (3+ buffers, one viewing transformation)
  - Reflective shadow maps (ok, this is still deferred rendering!)

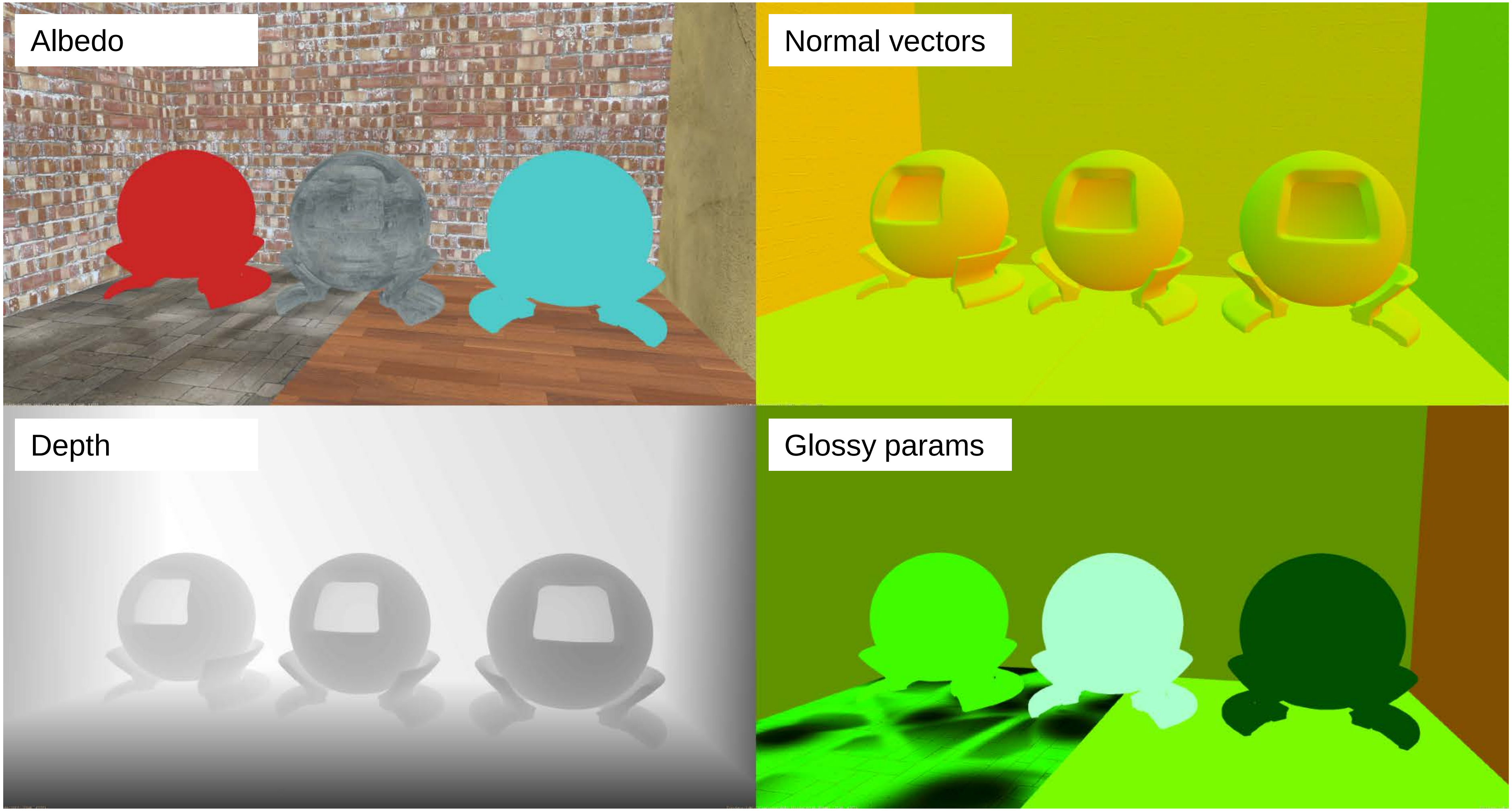
Multiple Render Targets (2)

- This is enabled via the Multiple Render Targets (MRT) mechanism:
  - The geometry is sent once for primitive generation
  - The pixel (fragment) shader writes results at the same location on multiple buffers
  - Different calculations and hence output values can be written to each buffer in the same pixel shader

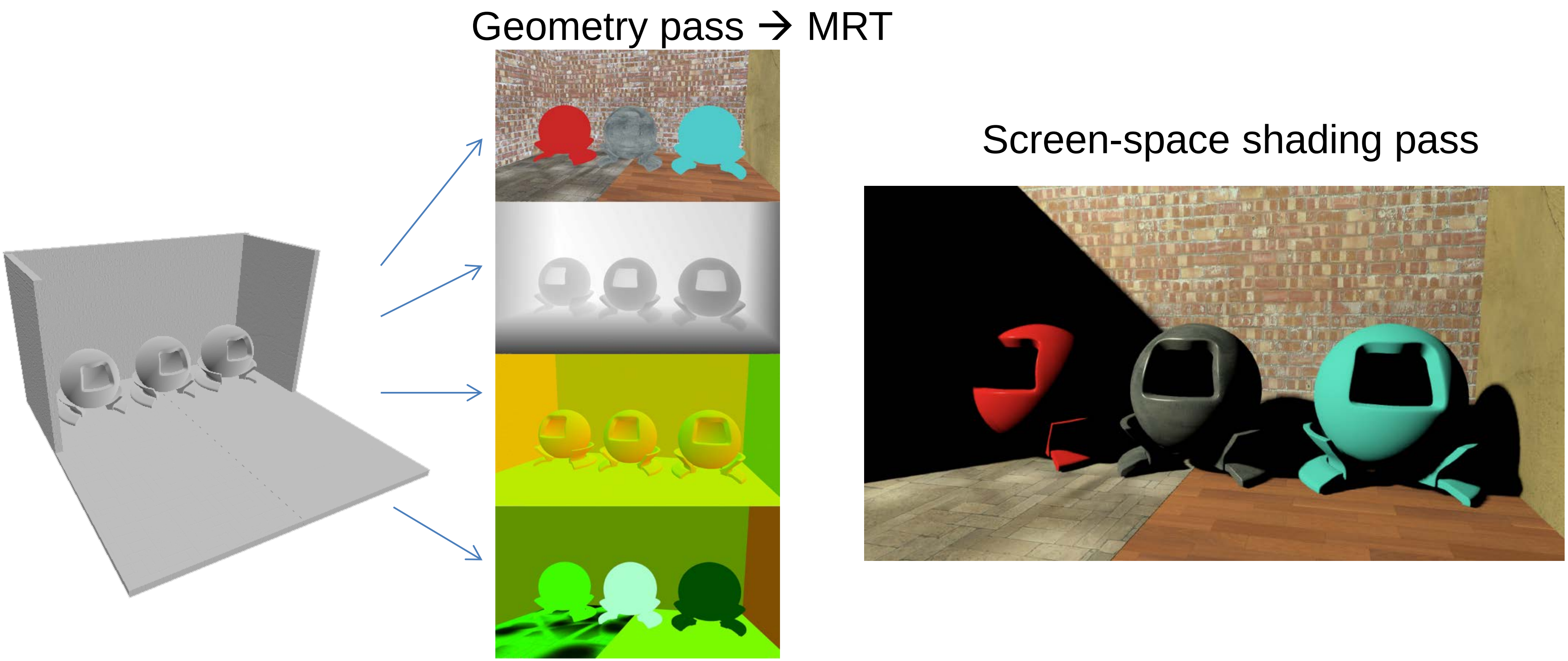
Multiple Render Targets (3)



Deferred Rendering - The Principle



Deferred Rendering - The Principle



- Instead of shading the fragments of each individual triangle in isolation, compute the final color for the resolved, visible geometry only

Source: AUEB Graphics Group XEngine

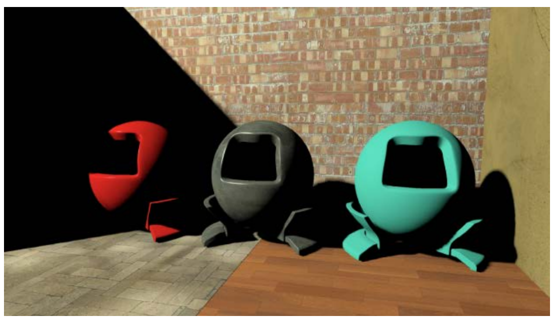
Deferred Rendering – Pros

- Geometry is rendered once, regardless of number of lights
- Shading rate is proportional to image size and NOT the amount of rendered geometry or depth complexity
  - Predictable, controllable and stable
- Capable of handling many more light sources
- Simplification of rendering pipeline

Deferred Rendering – Pros

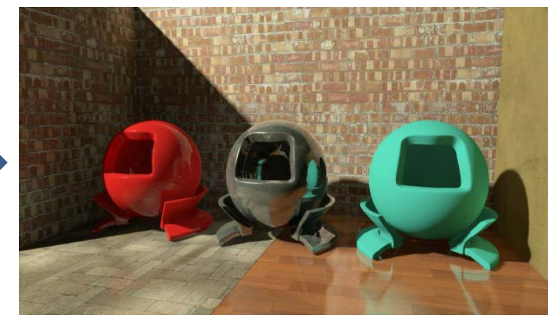
- Lighting algorithms and other rendering passes have access to global image data, not only the current fragment (e.g. see GI)

Direct



+Diffuse GI

+Specular GI



Radiance caching

Screen-space reflections

Deferred Rendering – Cons

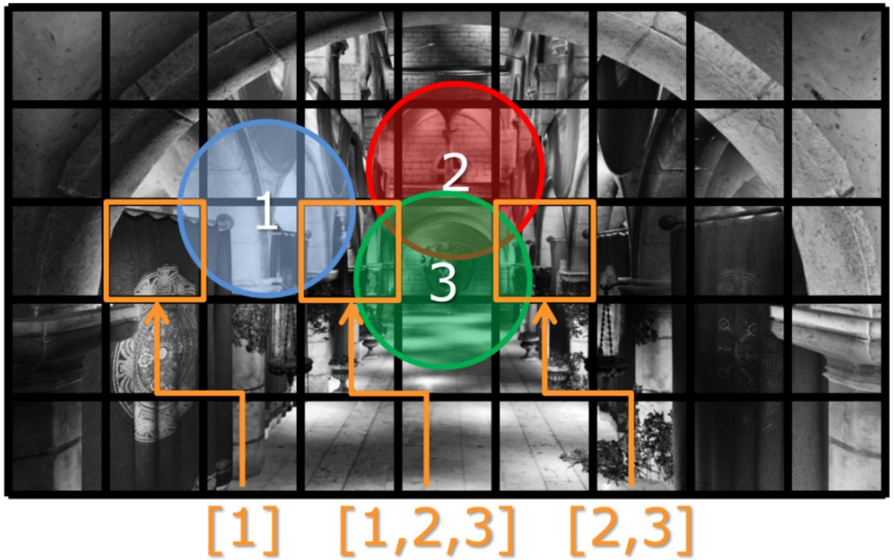
- Cannot handle transparent geometry. Still need a separate (forward) pass fro such surfaces.
- Does not mix well with antialiasing
  - MSAA pixel resolve requires a final color to be already available at the pixel samples. DR delays this computation
  - No sense in having MSAA filtered geometry attributes (not even correct).
- Limits the amount of different materials that can be used (requiring additional buffers to write their properties and IDs)

Tiled Rendering

- One problem with both forward and deferred rendering is the presence of a large number of light sources:
  - For each one, a lighting pass must be made OR
  - A large number of sources must be iterated within a loop in the fragment shader

Tiled Rendering

- Solution: Divide visible domain into tiles and assign light sources only to affected regions
- Prerequisite: each light source has a bounded area of effect (not really physically correct, but ok).



Clustered Rendering

- Tiling can also be done in the Z direction (clustered rendering):

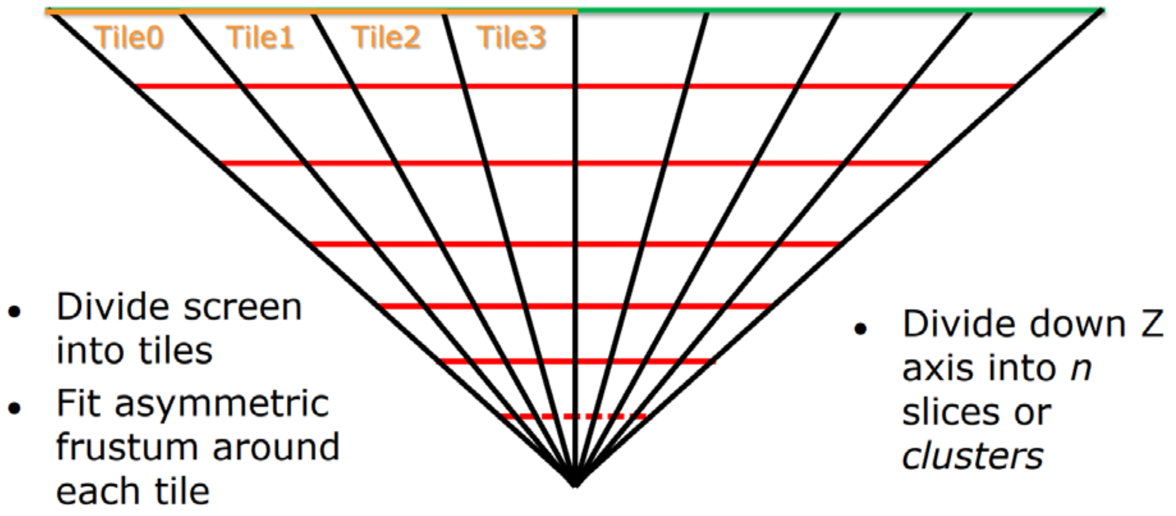


Image from: https://twvideo01.ubm-us.net/o1/vault/gdc2015/presentations/Thomas_Gareth_Advancements_in_Tile-Based.pdf

Clustered Rendering

- Clustered rendering also helps treat lights differently according to depth:
  - Fade them out
  - Drop back to glares
  - Prebake

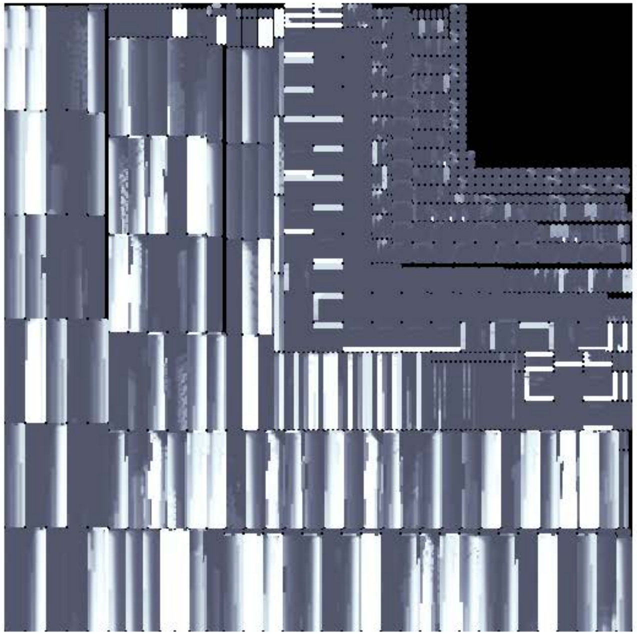
Image from: https://twvideo01.ubm-us.net/o1/vault/gdc2015/presentations/Thomas_Gareth_Advancements_in_Tile-Based.pdf

FAST APPROXIMATE LIGHTING

Light Maps

- Storage of pre-calculated (“baked”), view-independent illumination
- Store incident direct and/or indirect diffuse illumination in the texels of the map
- When object is rendered, the pre-recorded information on the light map is used, provided that:
  - Geometry is part of a static environment
  - Moving objects' contribution to diffuse illumination is negligible
  - Light-mapping is extensively used for the accelerated real-time rendering of realistic scenes
- Resolution of the light map does not need to be very high since illumination varies more slowly on a surface than a color or bump pattern

Light Maps



Texture Atlases

- A texture atlas is a surface parameterization where connected parts of the object's surface (*charts*), are each mapped onto contiguous regions of the texture domain
- Atlas ensures the unique mapping between Cartesian coordinates on the surface & locations on the bounded texture domain of the image map
- Construction:
  - Surface partitioning into charts
  - Unfold chart on a 2-D domain to ensure unique mapping
  - Pack chart parametric partitions into a single texture (NP-complete problem)

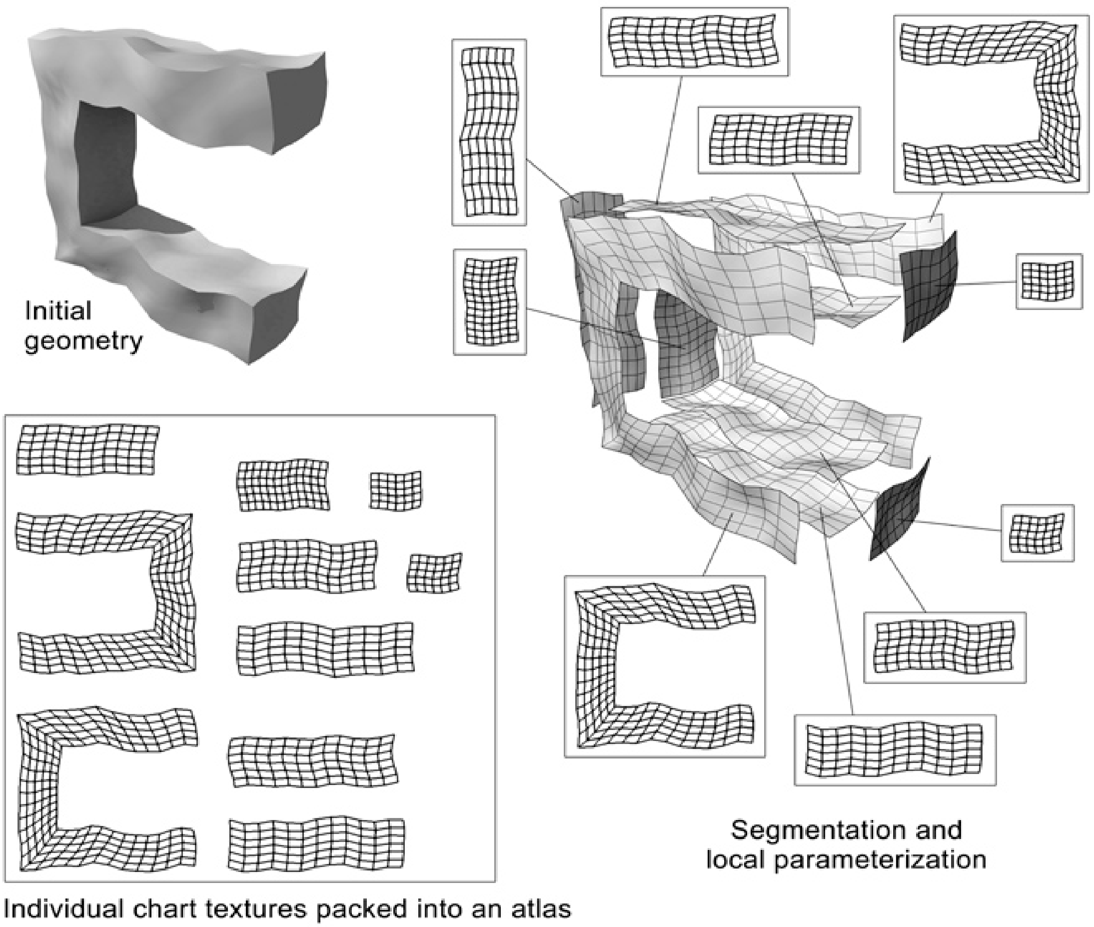
Texture Atlases

Criteria for chart partitioning and unfolding:

– Minimize texture distortion and artifacts
– Distribute the texels over the surface as evenly as possible
– Ensure continuity & conformity of mapping among the charts, if possible
– Maximize the area coverage of the charts & minimize the # of separate charts

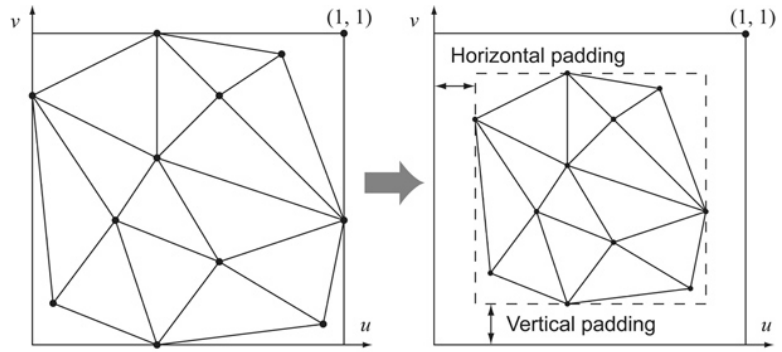
Texture Atlases - Polypacks

- Common and simple approach: polypacks
- Cut surface into regions (polypacks) and map each one to a plane with as little distortion as possible



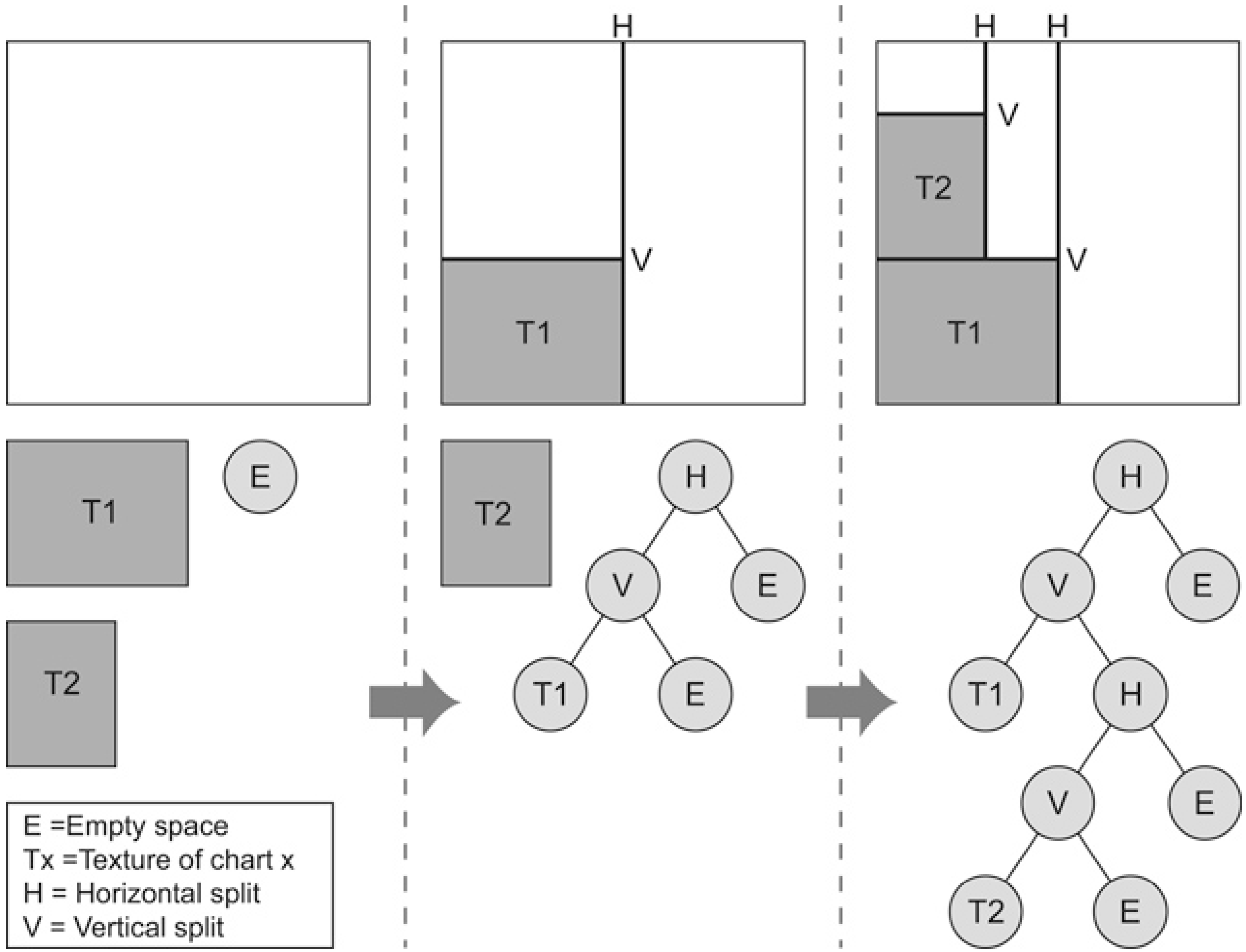
Atlas Generation Issues:

- As number of charts increases, so does the unused space:
  - Charts are not tightly packed to ensure some "guard space" between them to allow texel interpolation and mipmaping to work correctly



- Texel area coverage must be as close to uniform as possible:
  - Avoid stretching
  - Ensure proper and proportional scale of charts in packed atlas

Atlas Packing: kd-tree approach



Atlas Packing: horizon approach

- Suitable for large polygon charts with low compactness
- Operates in the discrete texture space
- Construction:
  - Rotate the charts so that their longest diameter is vertically aligned
  - Sort charts according to height and insert into the atlas
  - Incoming charts are stacked on top of the existing clusters in the atlas
  - Topmost texels occupied by the charts already in the atlas form a “horizon”, which the new chart's underside texels (“bottom horizon”) cannot penetrate

Lightmap Computation

- Lightmap texels are uniquely mapped to triangle locations and their attributes
- Iterate over valid lightmap texels
  - Compute lighting in texture space

- At runtime, transfer lighting onto shaded triangle fragments via texture mapping

Practical Lightmaps in Games

- Complex geometry limits the efficiency of lightmap packing
- Use simpler "proxy" geometry for lightmap calculation
- Map proxies to corresponding polygon groups
- Transfer proxy lighting onto detailed geometry

Practical Lightmaps in Games

Source: https://media.contentapi.ea.com/content/dam/eacom/frostbite/files/gdc2018-precomputedgioballilluminationinfrostbite.pdf

Complex Light Sources

- Large and complex-shaped emitters are challenging in real time:
  - Cannot use MC integration effectively in the time constraints of a real-time engine
- Typical useful emitters: spheres, quads, tubes
- Resolve to:
  - Analytical approximations for diffuse BRDFs
  - Image-based solutions for glossy/specular BRDFs and ray tracing

Area Lights – Diffuse BRDFs

- For a convex light source and a diffuse surface, the contribution of a light source boils down to computing irradiance from the projected visible surface (e.g. disk for a sphere):

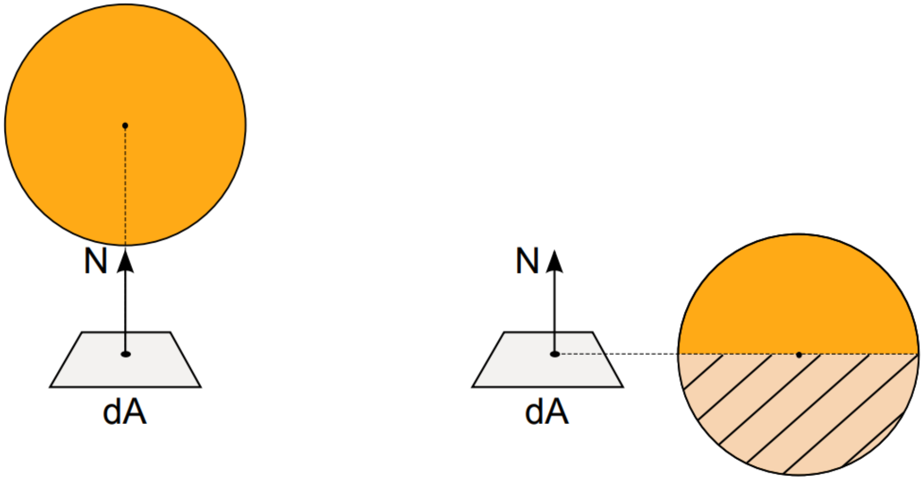
$$L_{out} = \frac{\rho}{\pi} \int_{\Omega_{\text{light}}} L_{in} \langle \mathbf{n} \cdot \mathbf{l} \rangle = \frac{\rho}{\pi} E(n)$$

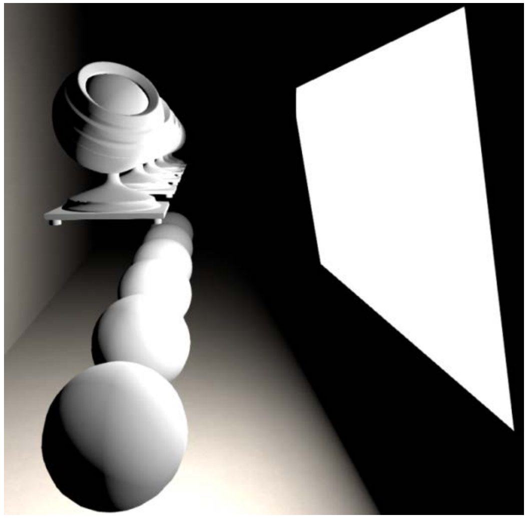
$$E(n) = \int_{\Omega_{\text{light}}} L_{in} \langle \mathbf{n} \cdot \mathbf{l} \rangle \, \mathrm{d}\mathbf{l} = \int_{A} L_{in} \frac{\langle \mathbf{n} \cdot \mathbf{l} \rangle \langle \mathbf{n}_a \cdot -\mathbf{l} \rangle}{distance^2} \, \mathrm{d}A$$

$$= L_{in} \boxed{\int_{A} \frac{\langle \mathbf{n} \cdot \mathbf{l} \rangle \langle \mathbf{n}_{\mathrm{a}} \cdot -\mathbf{l} \rangle}{distance^2} da} = L_{in} \quad \text{FormFactor}$$

Area Lights – Diffuse BRDFs

- The Form Factor integral can be approximated using MC samples or analytically estimated. However:
- The drop of the source below the horizon of the surface must be carefully handled!
- Sample representative points on emitter to compute FF.





Area Lights – Glossy BRDFs

- Necessary but crude approximation
- Treat all light coming from the emitter as coming from a single representative point on its surface
- A reasonable choice is the point with the largest contribution
- For a Phong distribution, this is the point on the light source with the smallest angle to the reflection ray

- Only reasonably good for emitters above the horizon
  - Apply some form of attenuation to handle horizon

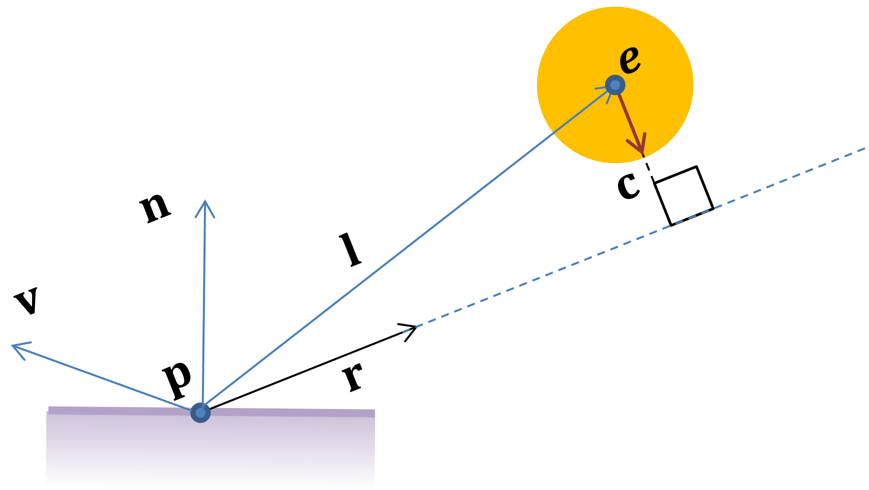
Area Lights – Glossy BRDFs

- Example: Spherical sources

$$\mathbf{c}' = \mathbf{r}(\mathbf{l} \cdot \mathbf{r}) - \mathbf{l}$$

$$\mathbf{c} = \mathbf{e} + \frac{\mathbf{c}'}{\|\mathbf{c}'\|} radius$$

$$\mathbf{r}' = \frac{\mathbf{c} - \mathbf{p}}{\|\mathbf{c} - \mathbf{p}\|}$$

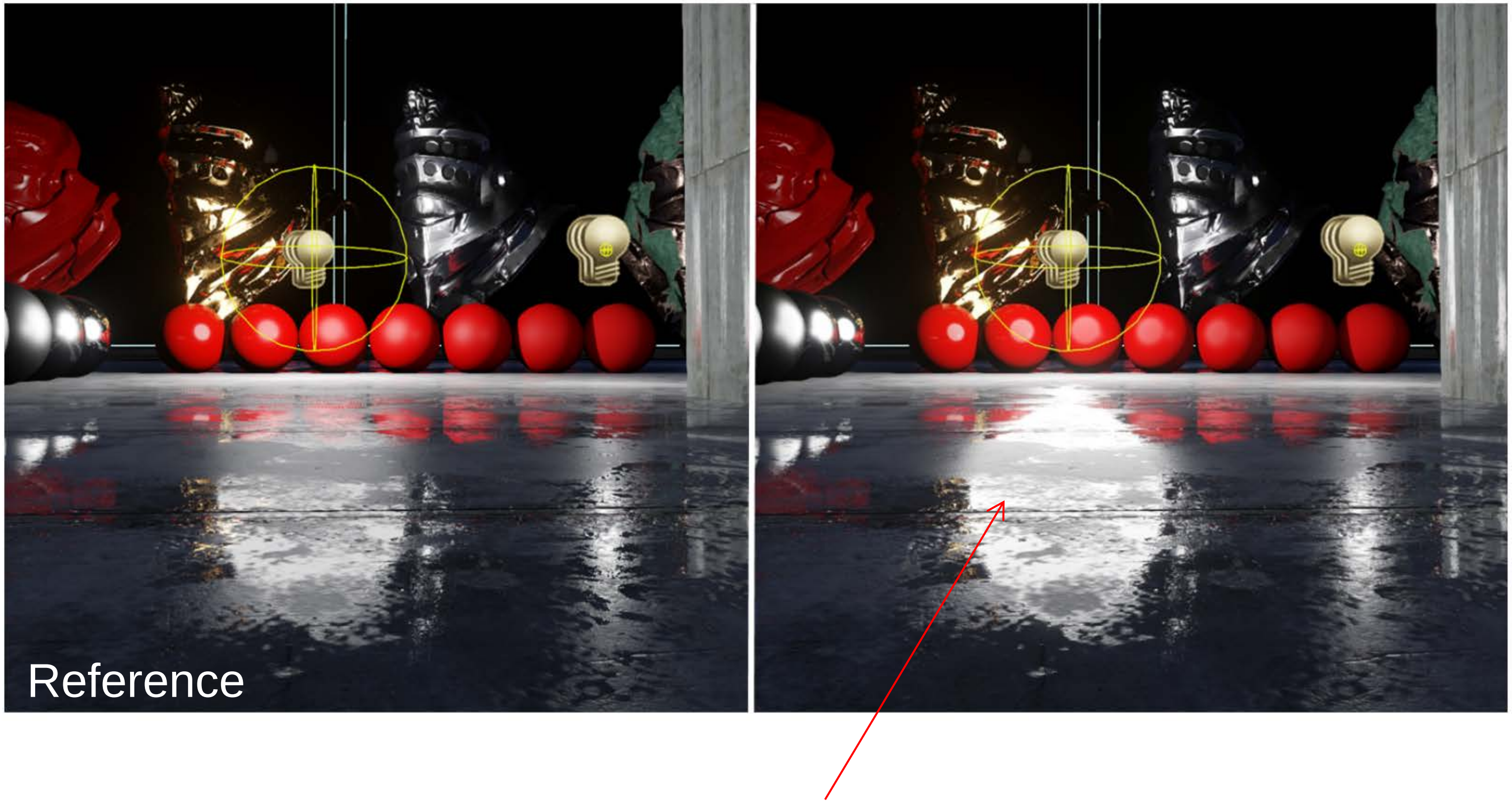


r: ideal reflection vector

l: **e** – **p**

Use this vector to light for shading

Area Lights – Glossy BRDFs



- Modified NDF requires normalization (too bright here)

Area Lights – Glossy BRDFs

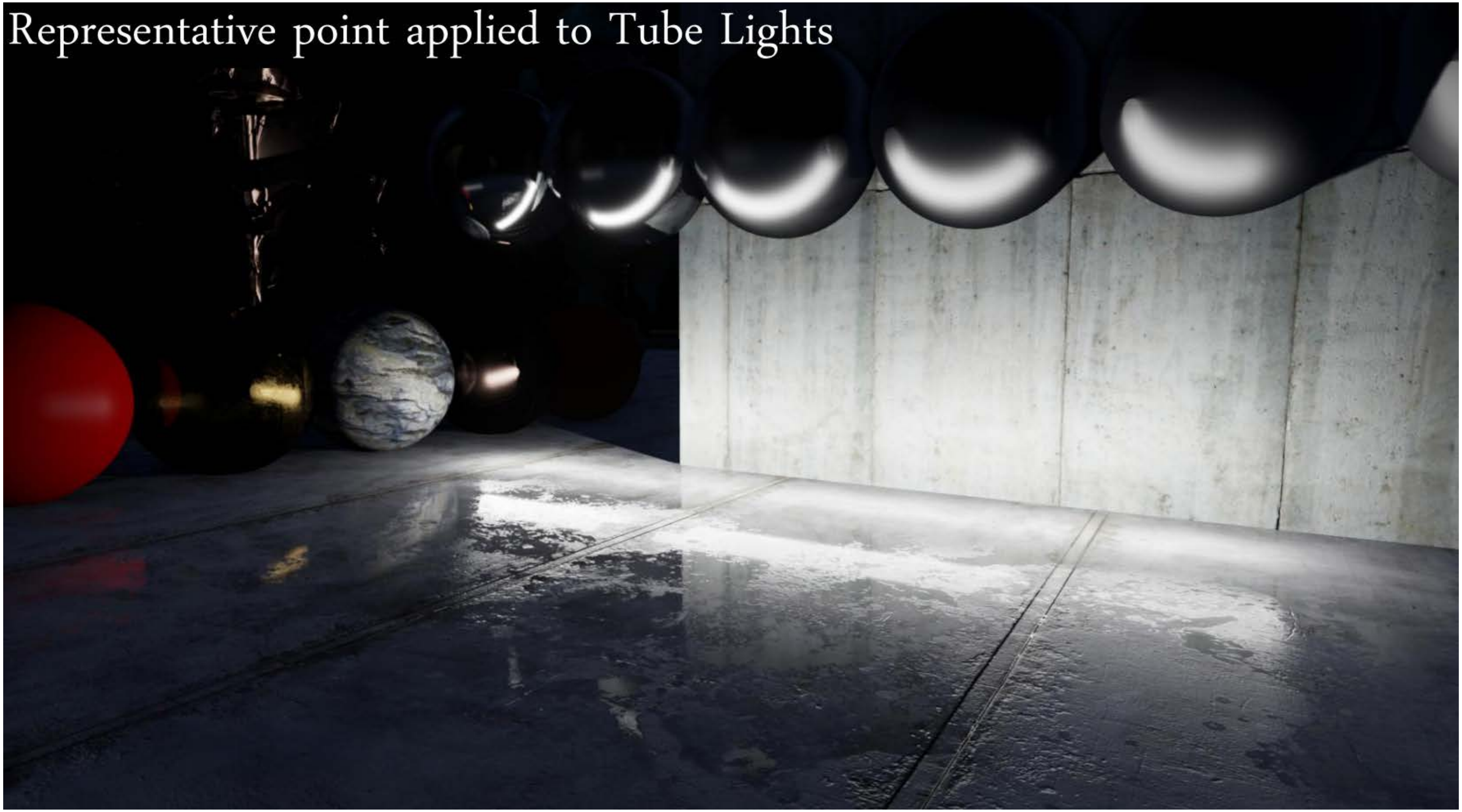


Image source: https://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_slides.pdf

IMAGE-BASED LIGHTING

Why image-based lighting?

- Very important in CG
- Helps transfer complex distant lighting to surfaces very fast
- Helps a rendered image blend with a real surrounding
  - Mix synthesized and real imagery (films, games, AR)

Environment Maps

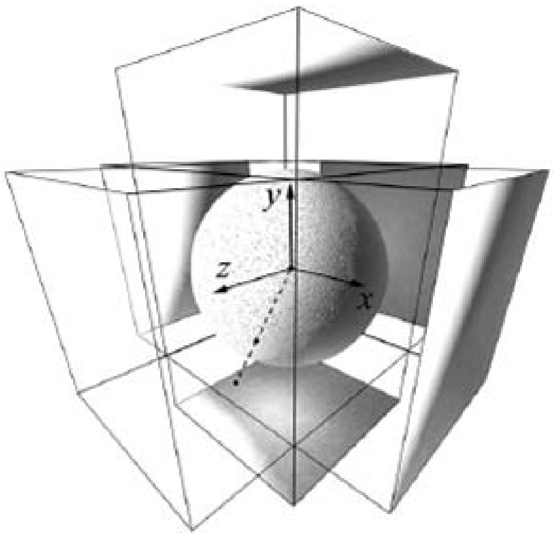
- An environment map is a representation of distant radiance parameterized w.r.t. an incoming direction ω_i
- Usually this information is discretely encoded on a set of images
- Other typical representations include spherical function coefficients

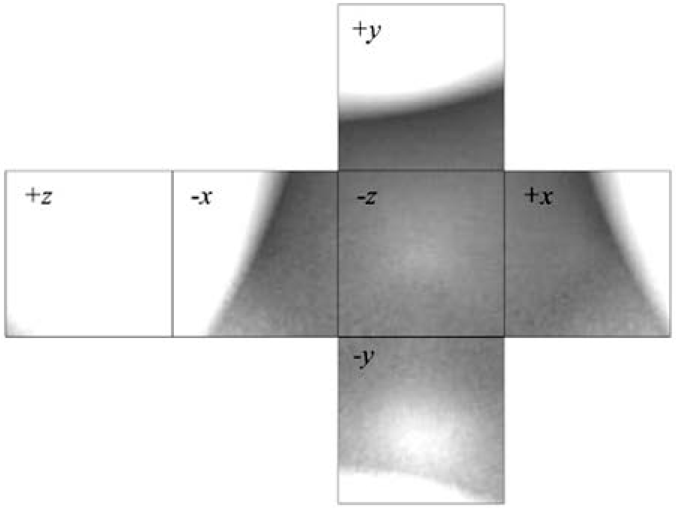
Environment Maps

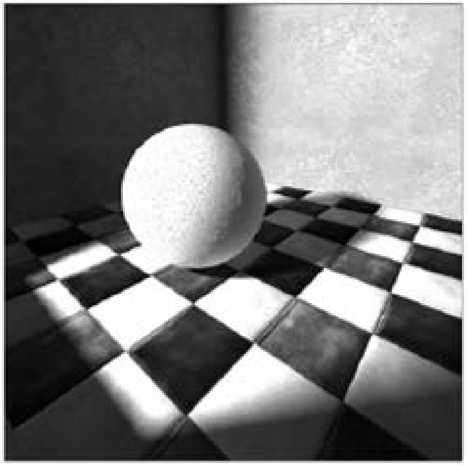
- Environment maps typically encode incoming illumination from the entire sphere around a point
- But can also be:
  - Hemispherical (e.g. sky lighting)
  - Cylindrical

Environment Maps

- Mostly in real time graphics, it is convenient to store the spherical environment in cube maps:







Light Probes

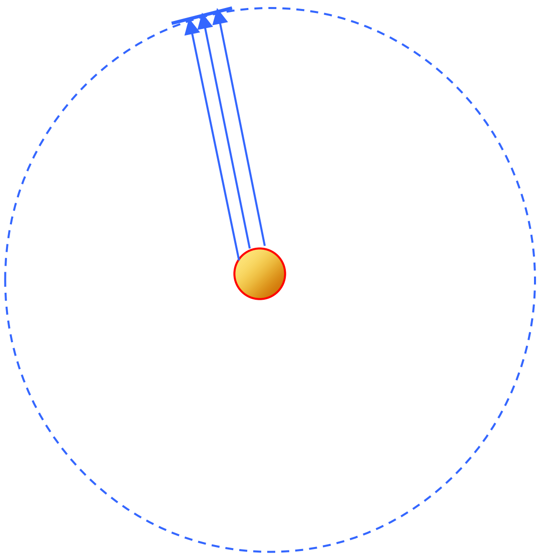
- Environment lighting images can be captured using physical light probes:
- Highly polished metallic spheres photographed to capture the real environment
  - Multiple exposures are typically taken to capture an HDR environment map

Light Probes

- To properly map the environment:
  - with low distortion and
  - Elimination of the photographic equipment from the image
- Multiple photos of the probe are captured
- The results are merged into an (inverse) panorama

Using an Environment Map

- The basic assumption about environment maps is that the environment is distant
- If assumed distant, incoming light is parameterized only by direction, as different points on the geometry will still index the same location on the environment map



Using an Environment Map

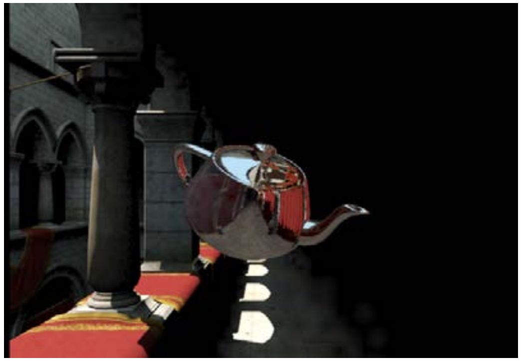
- Using as lighting the environment map on each point instead of using light sources:
  - Can provide a very natural look to artificial objects
  - Can blend the synthetic geometry with the captured environment
- This has been extensively used in movies and AR

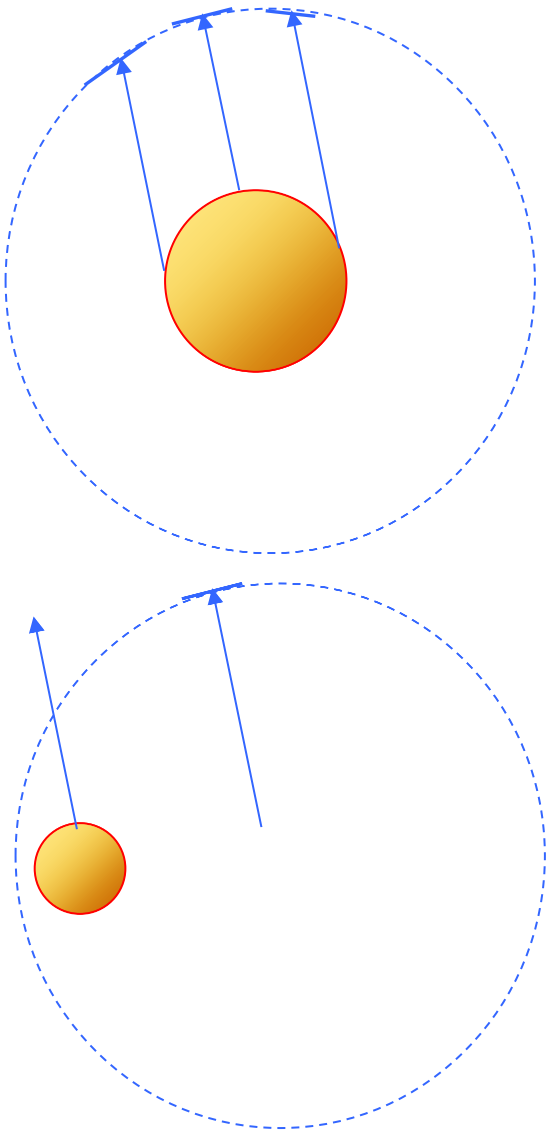
http://www.fxguide.com/featured/vfx-roll-call-for-the-avengers/

Using an Environment Map

- When environment distances are comparable to the size of the synthetic objects, a single environment map cannot do the trick

- Env. maps are also only valid for a particular region near the capture point





Virtual Light Probes

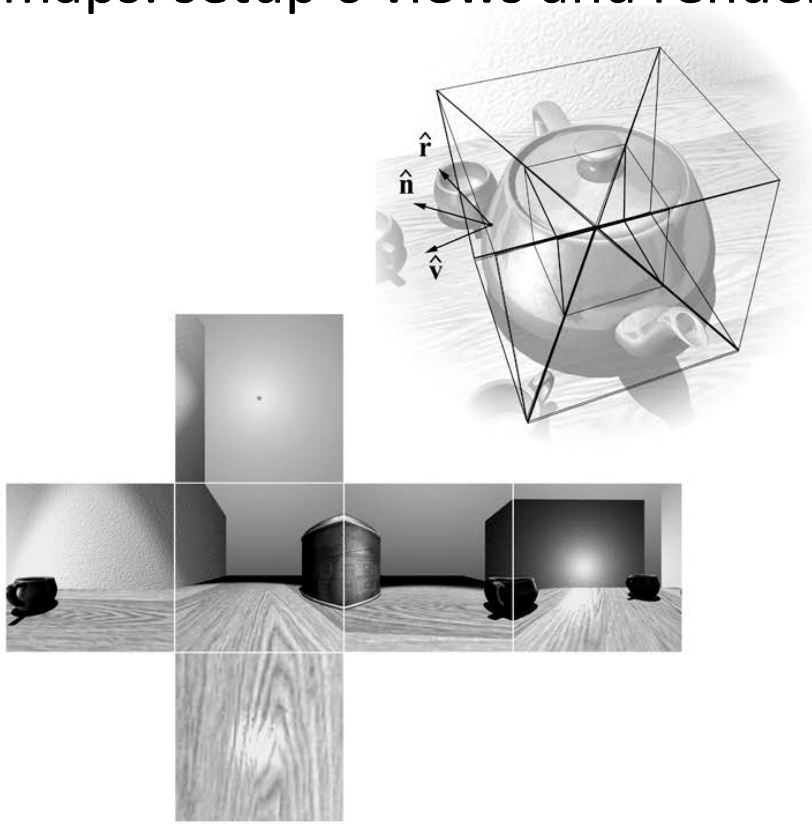
- In the previous example, the environment map was not captured from a real scene, but rather from a synthetic environment

Why do this?

- To significantly speed up indirect lighting calculations
- To apply indirect lighting to real-time rendering!
  - "Bake" incident light from a rendered environment
  - This lighting is the contribution of the env. Lighting to a surface
  - Can be combined with local shading from light sources

Virtual Light Probes

- Generation:
  - Via cube maps: setup 6 views and render the scene

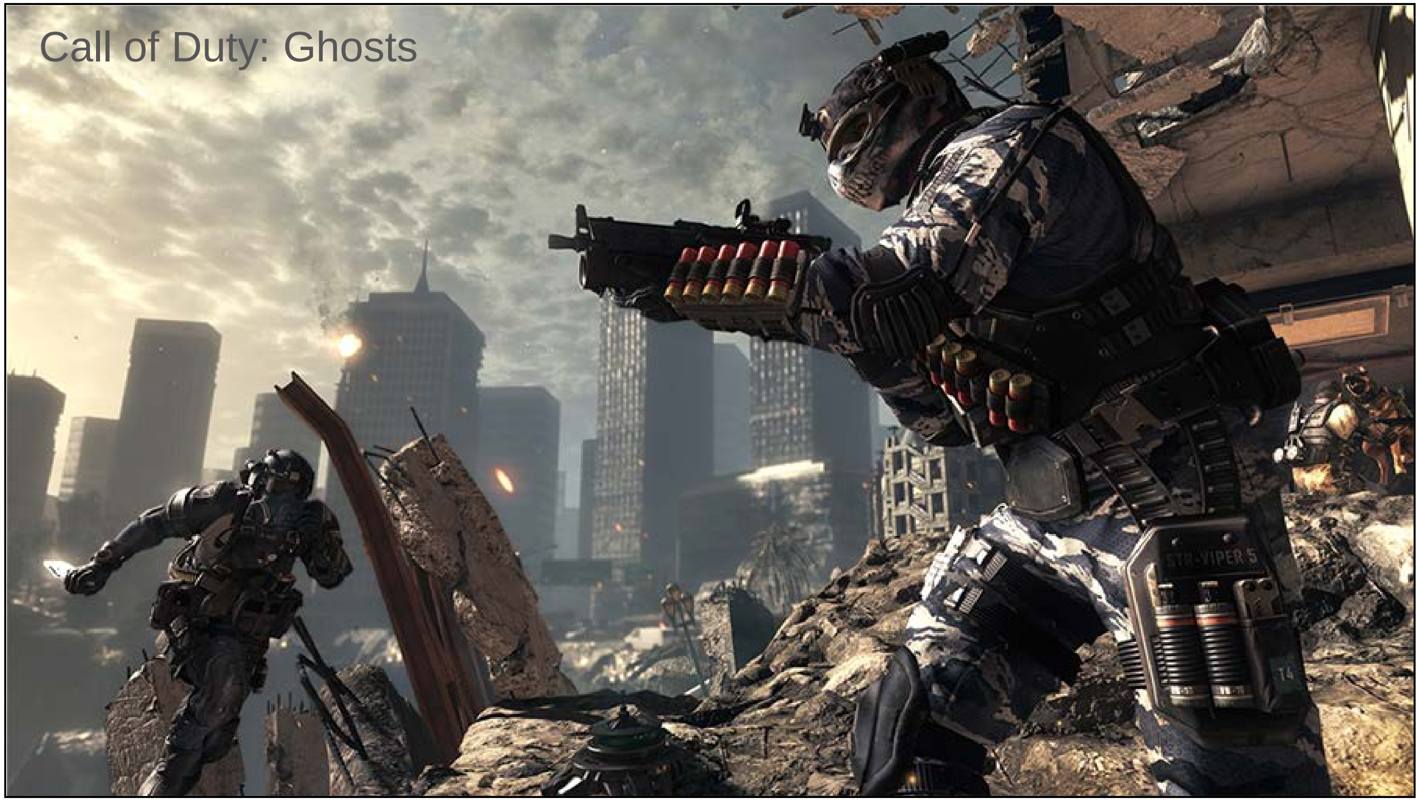


Virtual Light Probes

- Generation:
  - Directly sample the geometry and store a compressed spherical representation (see RT GI slides)

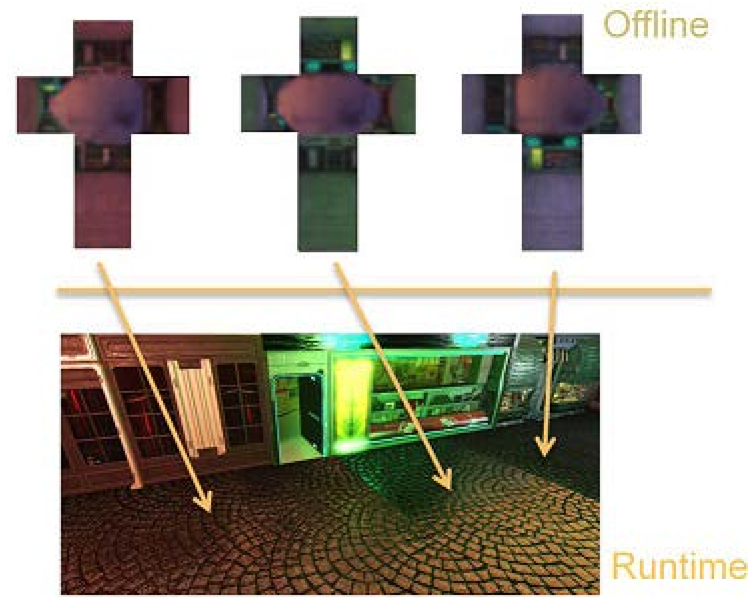
Environment Mapping in RT Applications

- Used for baking both rough indirect lighting and sky / ambient lighting



Multiple Light Probes

- To alleviate the invalidation of environment maps in different scene positions, multiple (virtual or physical) light maps can be generated from different locations
- At runtime, their contribution is interpolated



http://www.fxguide.com/featured/game-environments-parta-remember-me-rendering/

Irradiance Maps

- Environment maps encode the incoming light from a single direction ω_i
- So, in order to compute the reflected light on a surface, the contribution of all directions in the normal-aligned hemisphere must be accounted for, according to the reflectance equation:

$$L_o(\mathbf{x}, \omega_o) = \int_{\Omega_i} L(\mathbf{x}, \omega_i) f_r\left(\mathbf{x}, \varphi_o, \theta_{o,} \varphi_i, \theta_i\right) \cos\theta_i \, d\sigma(\omega_i)$$

- This is obviously computationally impractical in real time.

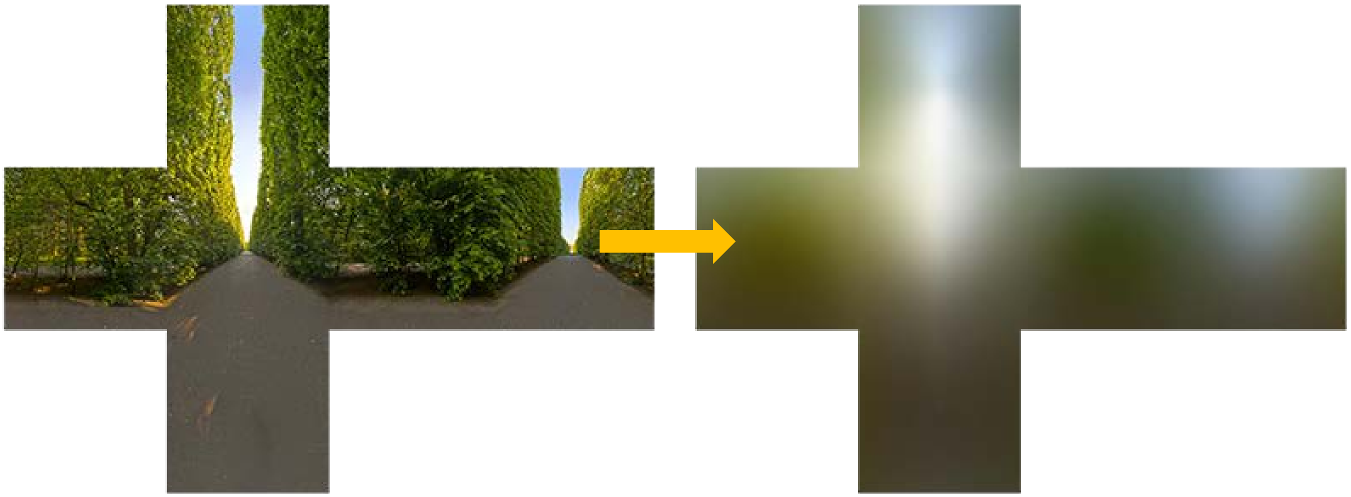
Irradiance Maps

- However, for the diffuse part of the BRDF, the integral can be greatly simplified:

$$L_o(\mathbf{x}, \omega_o) = \frac{\rho}{\pi} \int_{\Omega_i} L(\mathbf{x}, \omega_i) \cos\theta_i \, d\sigma(\omega_i)$$

- The integral has no dependence on ω_o and can be therefore pre-computed via MC integration with cosine-weighted IS for every possible hemisphere direction

Irradiance Maps



- Dropping the dependence on location (as in reflection maps), from the surface normal **n**:

$$L_o(\omega_o) = \frac{\rho}{\pi} IM(\mathbf{n})$$

What about Glossy BRDFs?

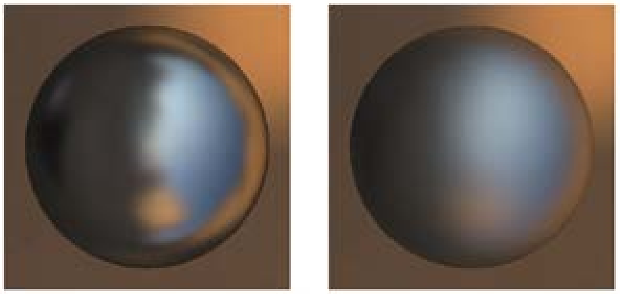
- The same cannot be done in the general case of glossy BRDFs, due to their dependence on ω_o
- However, if we consider that contributing directions are centered around the ideal reflection direction of ω_i , an approximate solution is possible:
- For different roughness values:
  - Precompute the irradiance inside a constricted solid angle centered at each ω_r direction, according to the spread of the BRDF
  - Store the versions as mipmaps of the same env. Map.

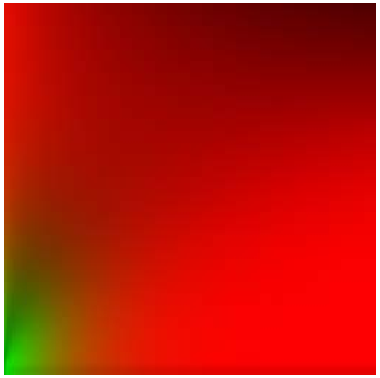
Pre-Convolved Environment Maps

$$L_o(\omega_o) = \int_{\Omega} L(\omega_i) f_r\left(\omega_o, \omega_i\right) \cos\theta_i \, d\sigma(\omega_i) \cong$$

$$\int_{\Omega_{lobe}} L(\omega_i) d\sigma(\omega_i) \int_{\Omega_{\mathrm{hemi}}} f_r\left(\omega_o, \omega_i\right) \cos\theta_i \, d\sigma(\omega_i) \cong$$

$$EM_{roughness}(\omega_r) \cdot M(\boldsymbol{\omega}_o \cdot \mathbf{n}, roughness)$$





Pre-Convolved Environment Maps

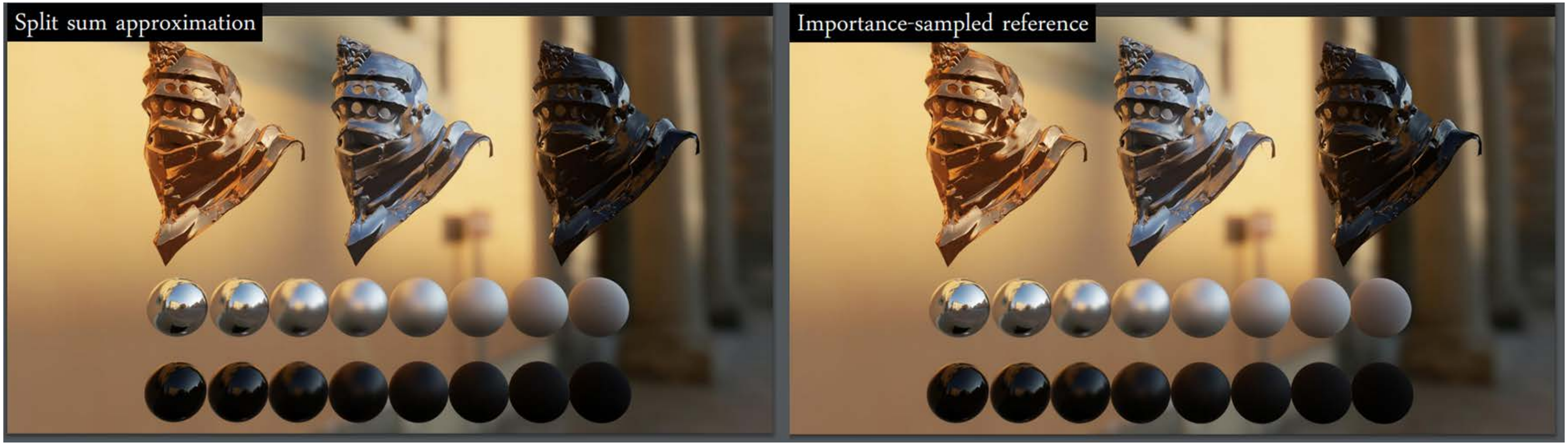


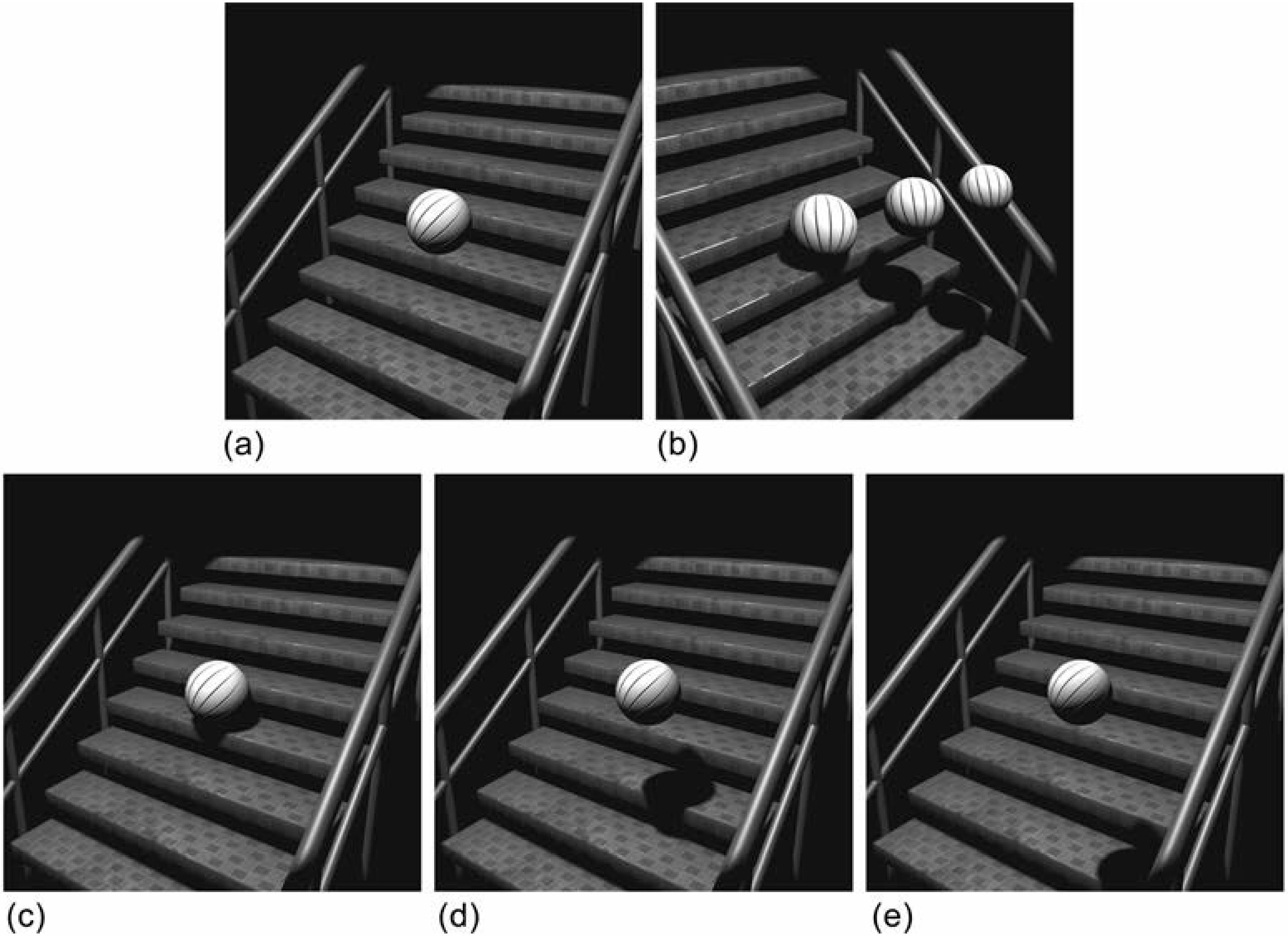
Image source: https://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_slides.pdf

VISIBILITY DETERMINATION

Shadows and Perception (1)

- Wherever there is light, there are shadows
- Presence of shadows:
  - Not only for aesthetic purposes
  - Provides clues for the shape of the geometry in the image
    - Helps place the objects in the environment. Gives clues about relative distances
    - Enhances depth perception: In monocular vision the HVS relies on clues and recognizable configurations to discern the ordering and distances of objects
    - Indicates the direction of incident light or light sources
- Enhances the visual detail of the displayed surfaces by enhancing local contrast

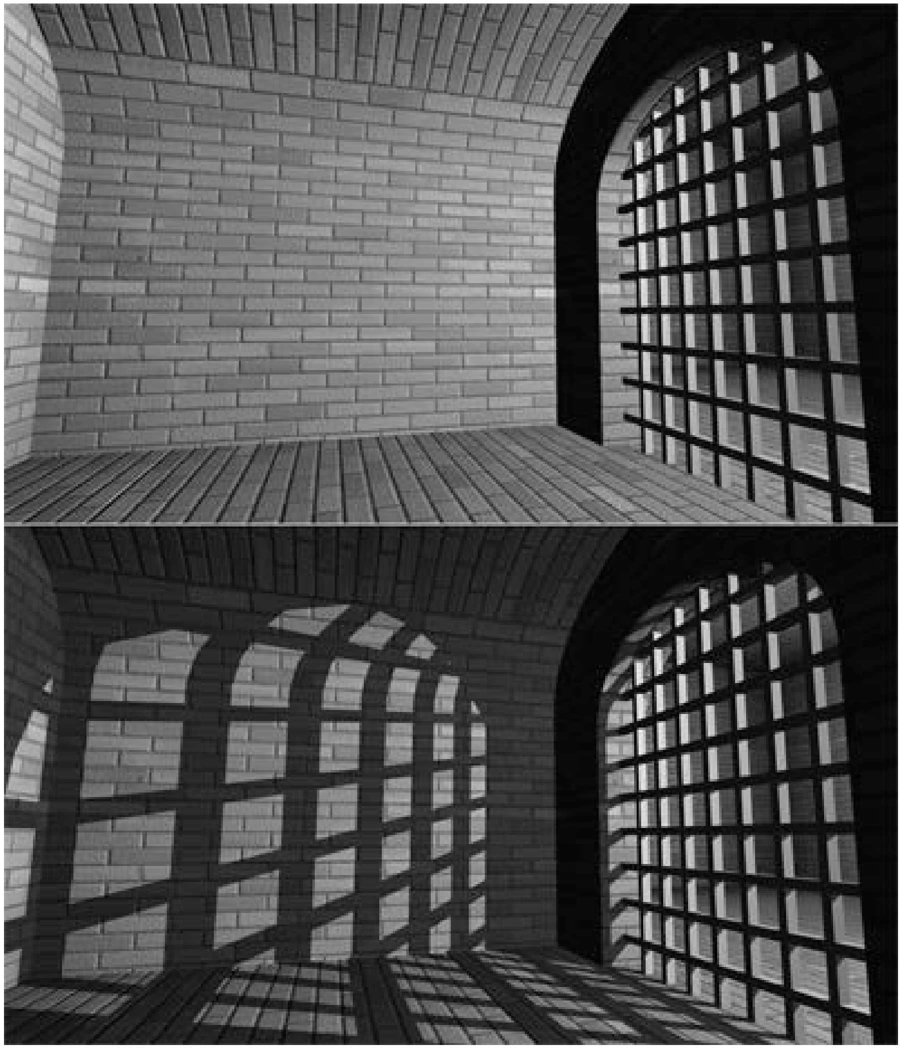
Shadows and Perception (2)


(a) (b) (c) (d) (e)

Shadows and Perception (3)

- (a) No shadow: We cannot possibly know the relative position or size of the ball w.r.t. the steps
- (b) Possible position/ball size configurations that lead to the same image (a)
- (c,d,e) The resulting images of the configurations in (b) when shadows are enabled

Shadows and Visual Detail



(no shadows)

Coarse, uninteresting surfaces

(with shadows)

Same geometry, higher visual detail

How are Shadows Generated?

- Partial or full obstruction of a source's light by geometry
- Indirect illumination reaching a surface is in general of lower luminance compared to the direct, unshadowed light →
- Illuminance of points in shadows is significantly lower than that of the lit points

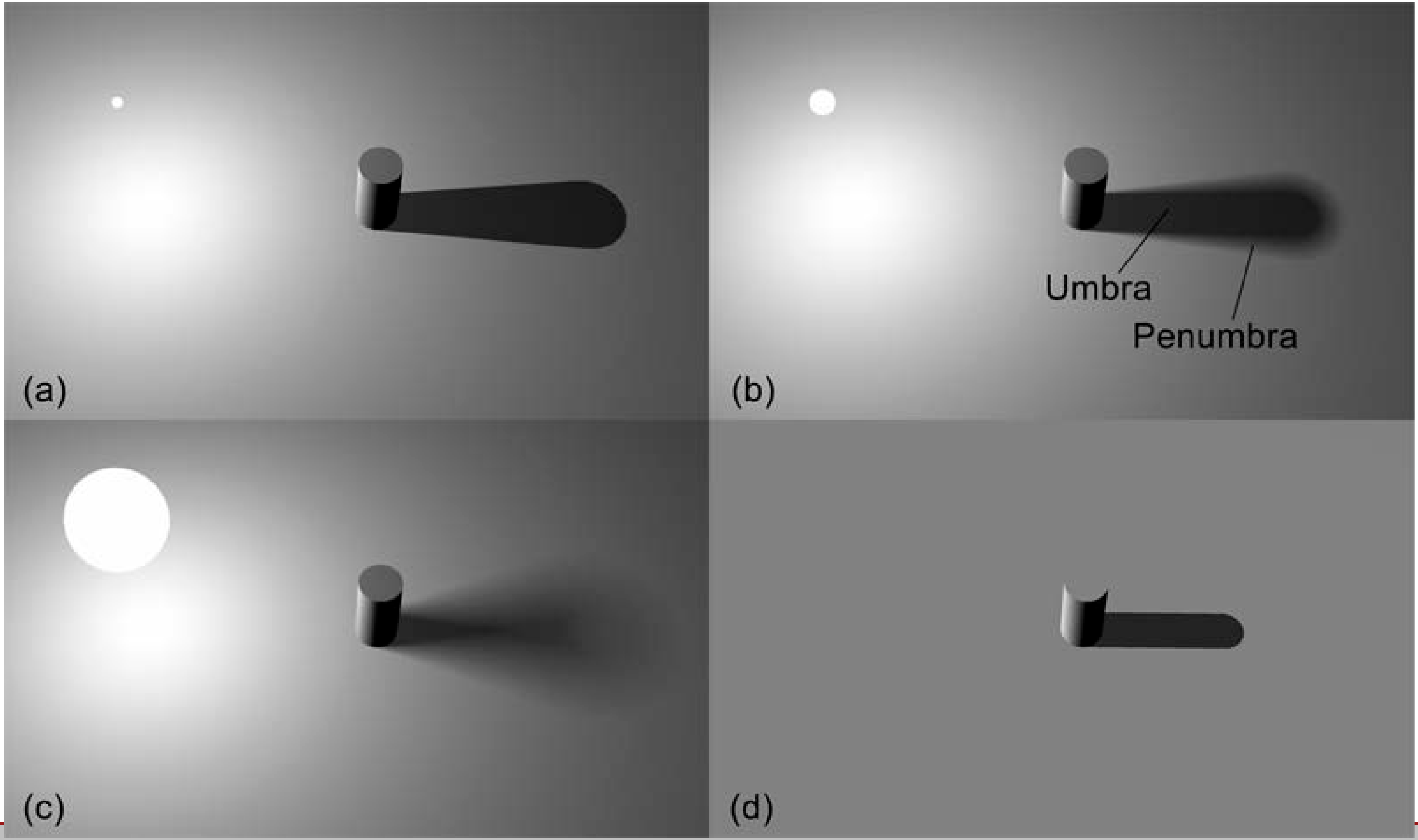
Shadow Types

- The size and type of shadows depend on the size and distance of the light emitting surfaces:
  - Infinitely distant light (directional) sources cause parallel shafts o shadows
  - Non-directional light sources cause radially projected shadow profiles

Umbra and Penumbra

- Umbra is part of the shadow due to complete light obstruction
- Penumbra is the shadow part where partial occlusion occurs and creates a soft transition to the lit surface (soft shadows)
- A punctual (point) light source creates hard shadows with no penumbra
- A light source with a non-negligible size and comparable distance to the occluding geometry causes shadows with penumbrae (soft shadows)
  - Larger emitters and smaller distances to occluders → larger penumbrae

Shadow Examples

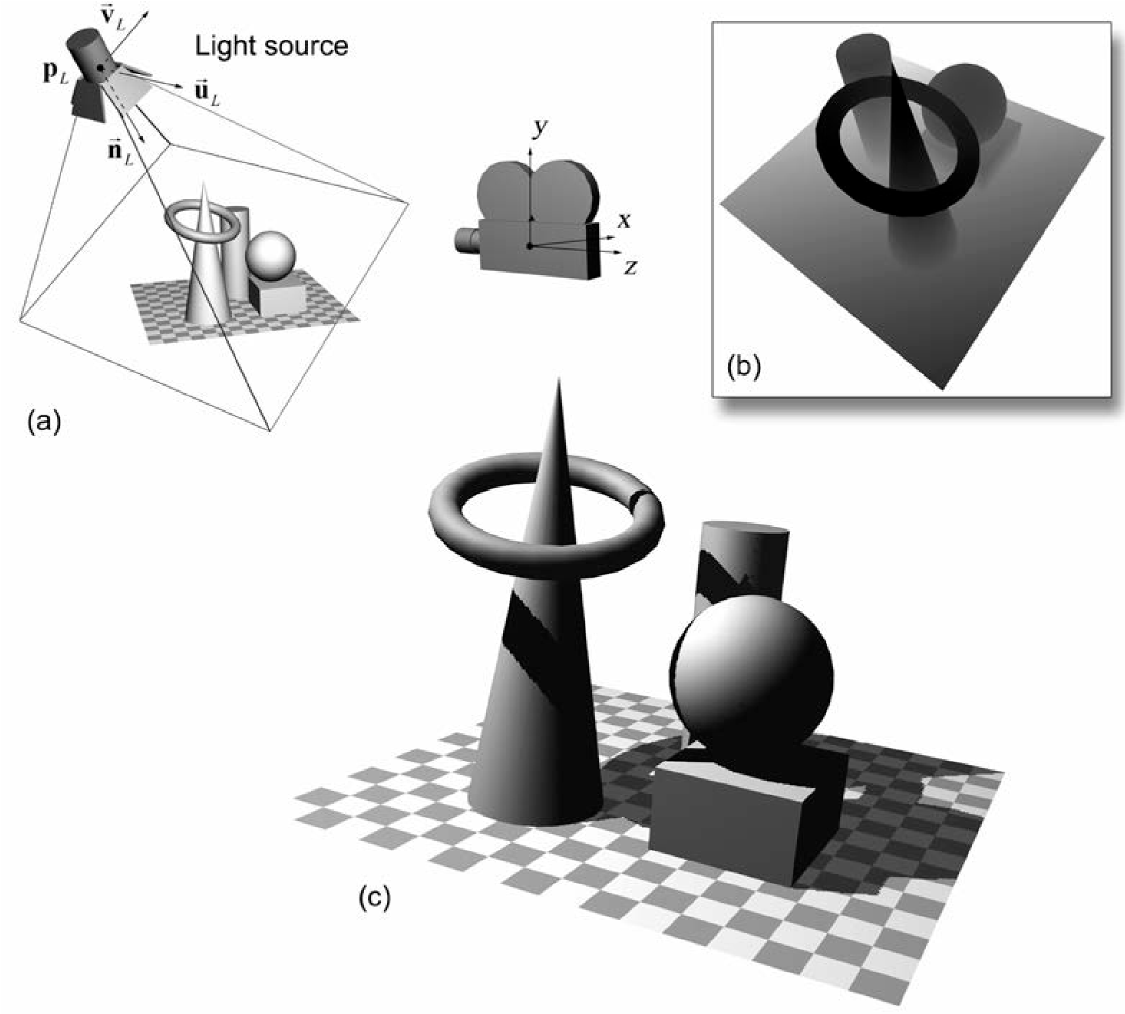


Shadow Maps

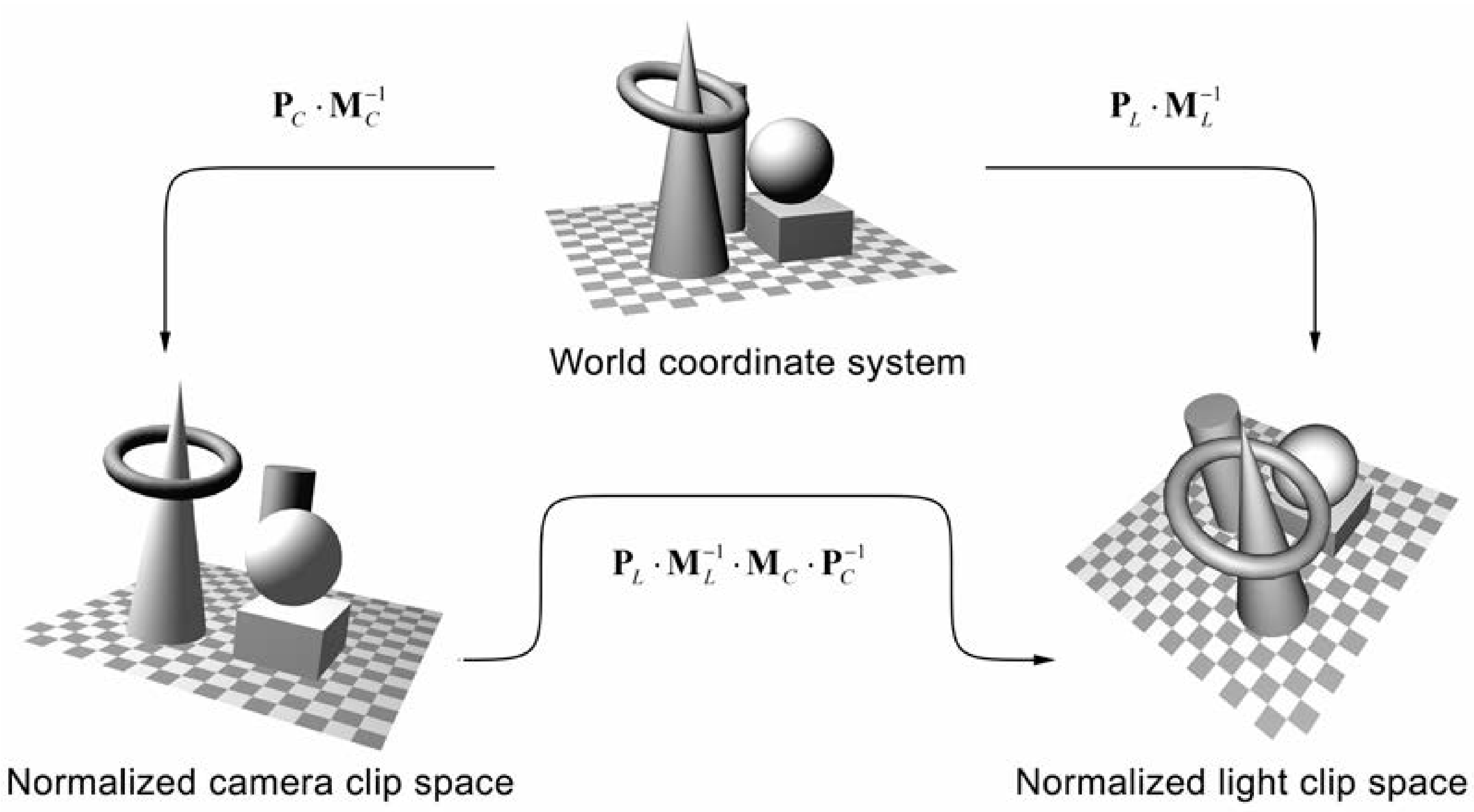
- Basic principle:
  - The occlusion of light on a surface due to a given (point) light source is a similar problem to the visibility determination from the user's view point
  - A point is lit if the point is the closest one to the light source in this direction, i.e. if it is "visible" from the light source
- We can use the depth buffer mechanism to perform HSE and determine the nearest visible points from the light source's view point
- We call the depth buffer generated from the light source view point a shadow map

Shadow Map - Setup

- A projection is set up from the light source's point of view (a) and the shadow map is captured (b)
- The scene is rendered normally form the camera view point and fragments are tested against the shadow map (c)



Transforming Fragments to S.M. Space



Shadow Calculations

- Render the scene from the light source view point $(\mathbf{p}_L, \vec{\mathbf{u}}_L, \vec{\mathbf{v}}_L, -\vec{\mathbf{n}}_L)$
  - Transform geometry by $\mathbf{P}_L \mathbf{M}_L^{-1}$
  - Record the depth (shadow) map Z_L
- Render the scene normally, from the camera view point
  - Transform each fragment from the camera CSS to the light source's CSS:

$$\mathbf{p}'_{frag} = (x'_{frag}, y'_{frag}, z'_{frag}) = \mathbf{P}_L \cdot \mathbf{M}_L^{-1} \cdot \mathbf{M}_C \cdot \mathbf{P}_C^{-1} \cdot \mathbf{p}_{frag}$$

  - Compare the fragment's light space z'_{frag} value with the corresponding depth in the shadow map $Z_L(x'_{frag}, y'_{frag})$
  - If $z'_{frag} \le Z_L(x'_{frag}, y'_{frag})$ the fragment is lit, otherwise it lies in shadow

Shadow Maps – Remarks (1)

- The shadow map needs to be updated only if:
  - The light source is moving
  - Geometry within the light's field of view changes
- The shadow map rendering time is significantly lower than the normal rendering time:
  - Only fragment depth is captured
  - No pixel shading occurs (pass through shader), no color attachment

Shadow Maps – Remarks (2)

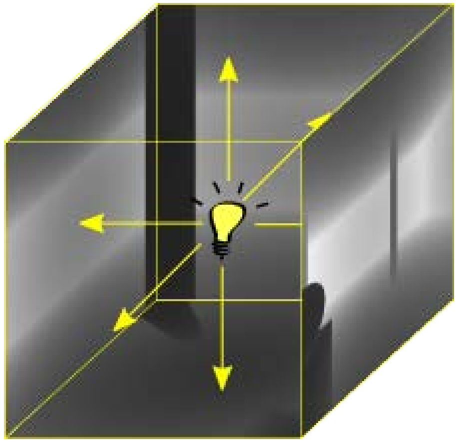
- WYSIWYG: Whatever geometric entity can be rasterized or otherwise drawn in a depth map, can be used as an occluder:
  - E.g. foliage modelled as polygons with transparent textures

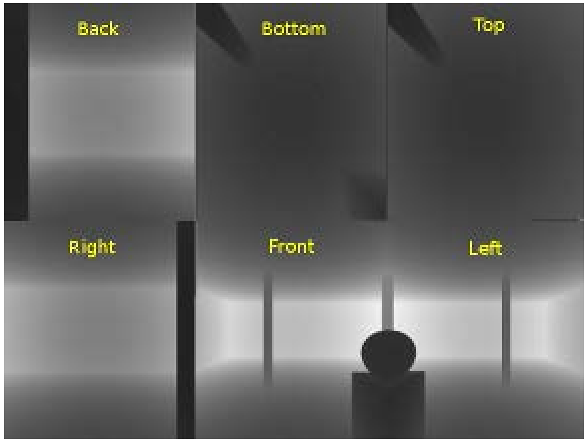
Advantages of Shadow Maps

- A simple and intuitive 2-pass algorithm
- Any renderable entity can generate shadow
- Easily combined with other effects, such as volumetric lighting
- Low complexity, takes advantage of GPU’s early culling mechanisms
- Linear dependence on scene complexity
- Adjustable SM size → performance/quality trade off
- Can generate soft shadows (via extra samples)

Shadow Map Problems (1)

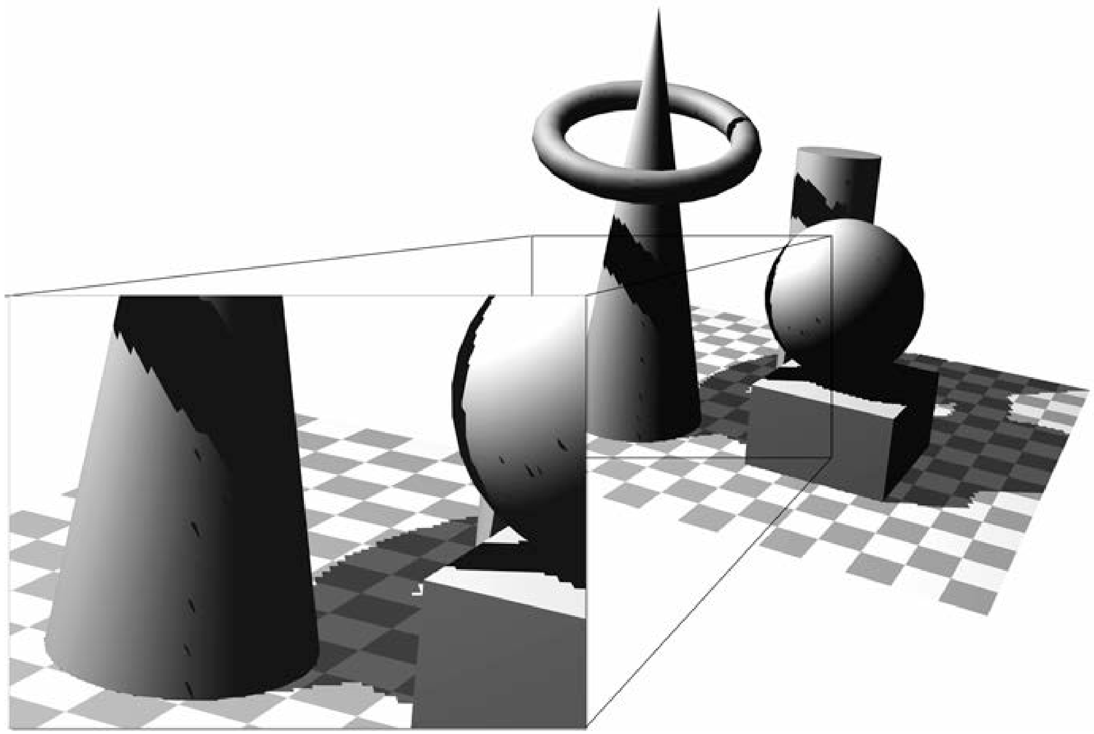
- Only works for conical/directional light sources
  - For omnidirectional lights, we need a cube map configuration of shadow maps



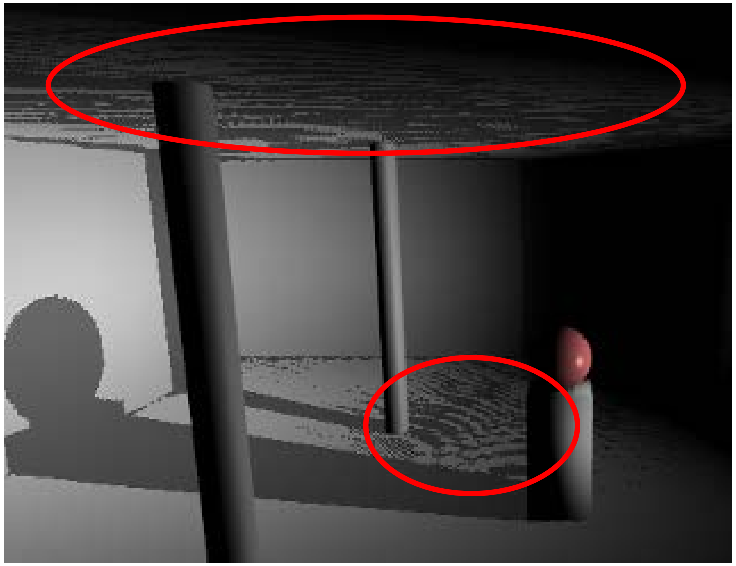


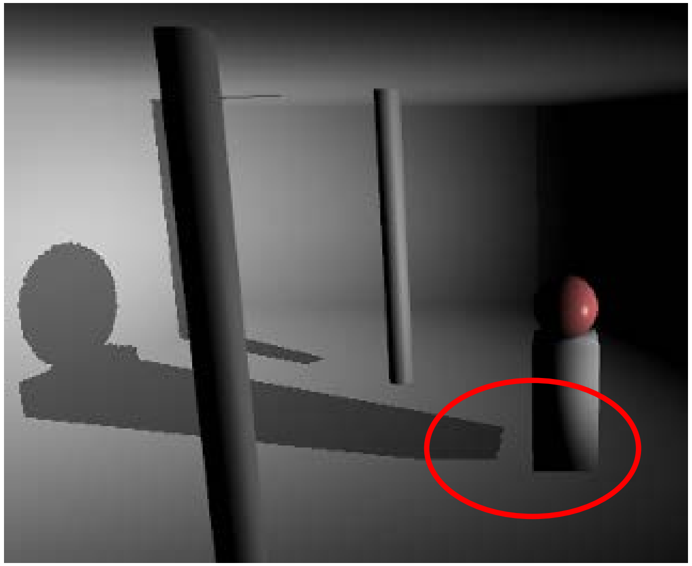
Shadow Map Problems (2)

- Accuracy depends on relative light-camera position and orientation
- Strong aliasing artifacts due to undersampling and arithmetic precision



Typical Shadow Map Artifacts





Shadow "acne"

"Peter Panning"

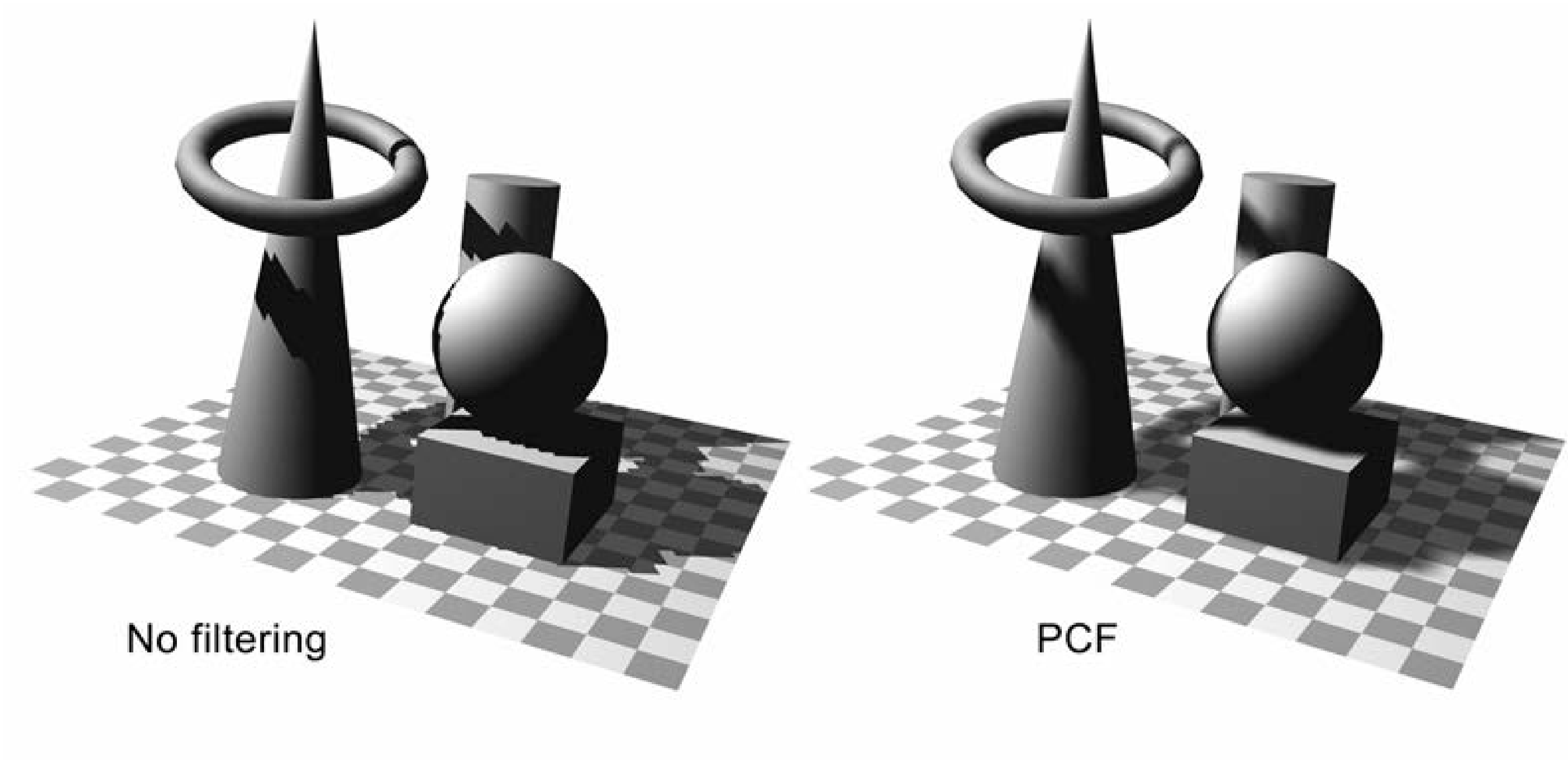
Shadow Map Antialiasing

- Typical bilinear filtering on the shadow map does not work
- If we pre-filter (mipmap) the shadow maps:
  - We filter depths! → Erroneous depth comparisons and we do not get rid of artifacts
- We need to change the order of filtering and comparisons: post-filtering

Percentage Closer Filtering

- Draw samples from the shadow map in the neighborhood of the query shadow map coordinate
- Individually test each shadow map tap with the fragment z
- Average the shadow test results to get the fraction of occlusion

PCF Shadow Maps Example

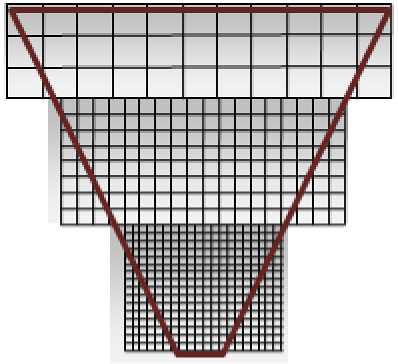


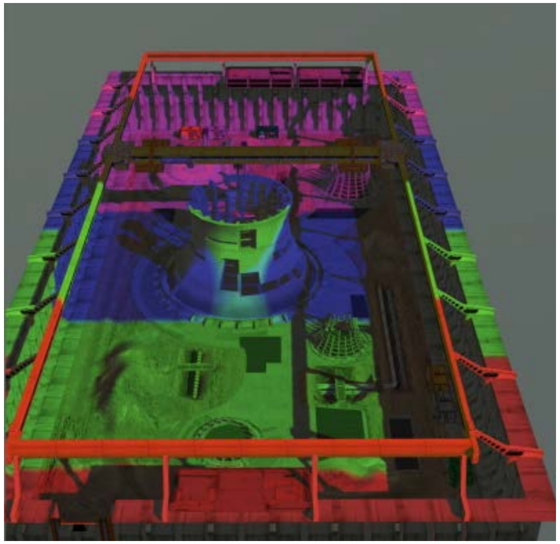
Cascaded Shadow Maps

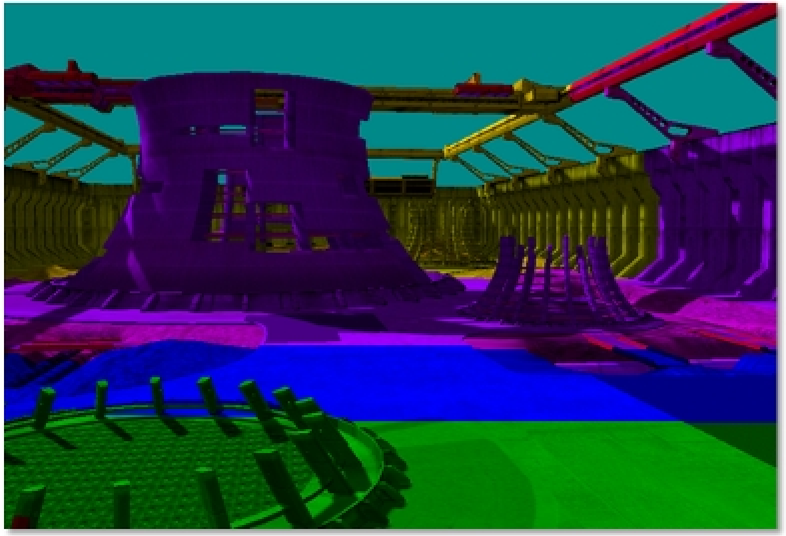
- Cascaded shadow maps (CSMs) are the best way to combat one of the most prevalent errors with shadowing: perspective aliasing
  - Different areas of the camera frustum require shadow maps with different resolutions
  - Objects nearest the eye require a higher resolution than do more distant objects

Cascaded Shadow Maps

- Basic idea:
  - Partition the frustum into multiple segments
  - A shadow map is rendered for each sub-frustum
  - The pixel shader samples from the map that most closely matches the required resolution







Cascaded Shadow Maps

- Typical setup:
- Multiple, same resolution cascades, but
- Covering an increasingly wider area
  - Decreasing fidelity away from user
  - Countered by perspective foreshortening
- Switch according to distance from user

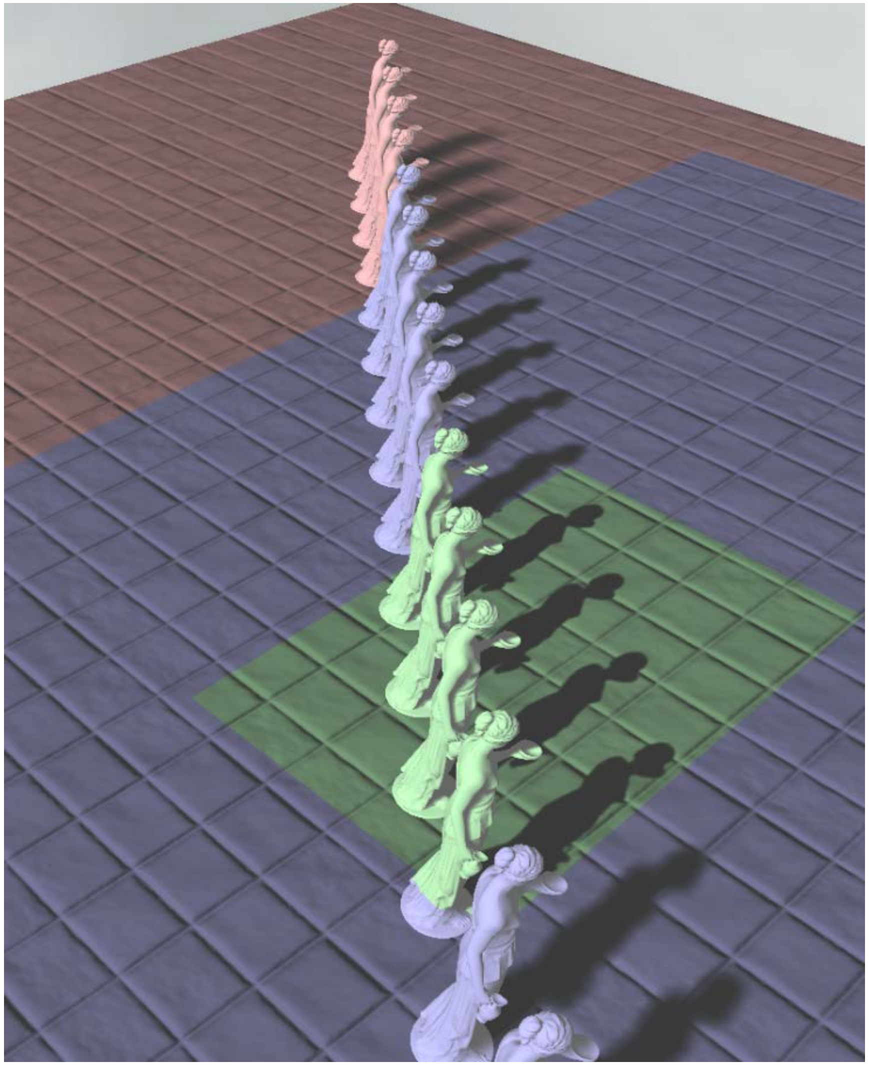


Image source: https://devansh.space/cascaded-shadow-maps

Cascaded Shadow Maps

- Construction:
  - Partition the frustum into sub-frusta.
  - Compute an orthographic projection for each sub-frustum.
  - Render a shadow map for each sub-frustum.
  - Render the scene.

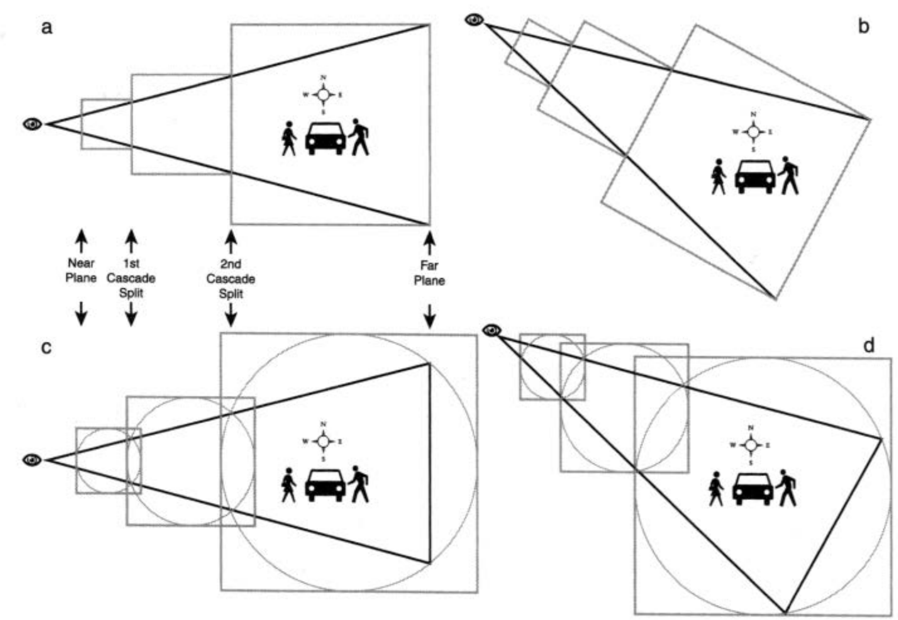
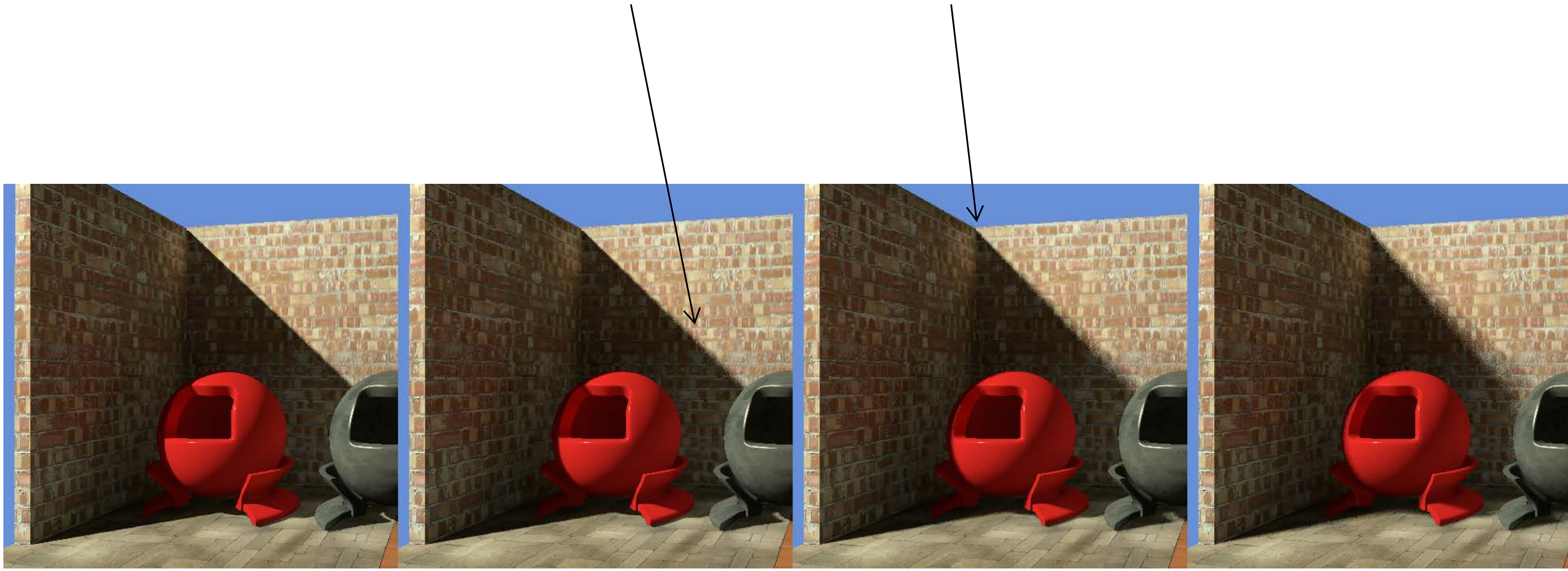


Image source: https://johanmedestrom.wordpress.com/2016/03/18/opengl-cascaded-shadow-maps/

Shadows from Area Lights

- Typically soft shadows are approximated by dynamically changing the PCF kernel size according to distance of occluded point from occluded geometry:
- $r_{PCM}(\mathbf{p}) = r_{PCM}(1 + \mathbf{p}_{ECS} - shadowmap_{ECS})$



Per-object Shadow Maps

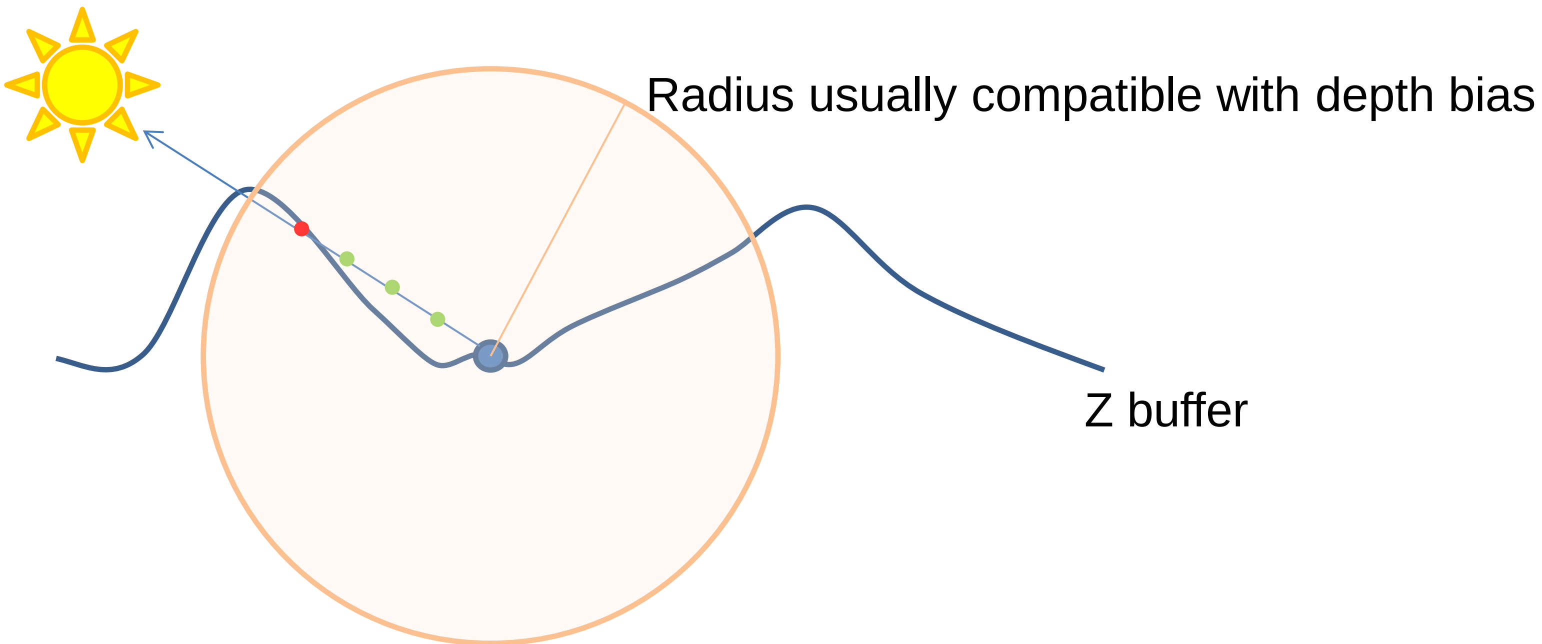
- Shadow maps can be focused also on certain high impact (e.g. close to the user) objects
- Dedicated SMs that are used for specific objects, instead of the global SMs or CSMs

Image source: https://www.cryengine.com/features/view/visuals

Screen-space Self-shadowing

- Screen-space shadowing is introduced to alleviate problems of shadow maps due to:
  - distance bias used for correcting shadow acne problem
  - Low resolution of SMs at close object inspection
- Idea:
  - March a ray (take samples on a short distance on the direction) from the shaded point towards the light source
  - Check for occlusion with depth buffer
  - Requires deferred shading

Screen-space Self-shadowing



Shadows

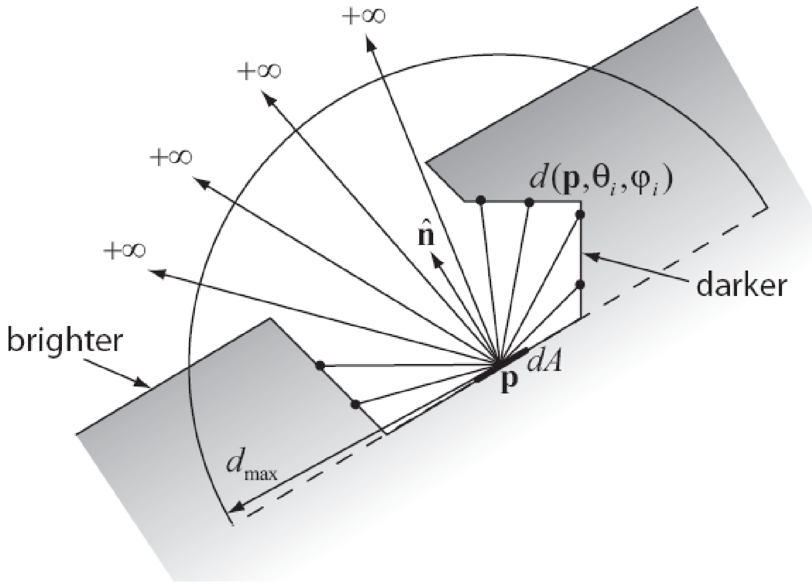
- Raytraced shadows
- Shadows from area lights
- Contact shadows
- Ambient occlusion
- Transparency

Ambient Occlusion

- A cheap way to simulate contribution of ambient (global) lighting
  - Though only convincing for outdoor scenes mostly
- Accentuates crevices → increases image contrast
- Estimates the overall drop of irradiance on the shaded point from occlusion due to near-field geometry

Ambient Occlusion Estimation

- Local or global illumination model?
- Hybrid!
  - Does not exchange light with other locations
  - Potentially search for occlusion up to a distance
  - Still requires visibility checks → intersections with other geometry



Ambient Occlusion Estimation (2)

- The value of occlusion shading can be easily determined if we set L_i in the reflectance equation to 1 and replace visibility with an attenuation score:

$$w(\mathbf{p}) = \frac{1}{\pi}\int_{\Omega} \mu\big(d(\mathbf{p},\omega_i)\big)d\sigma_{\perp}(\omega_i)$$

- Where $d(\mathbf{p},\omega_i)$ is the distance to the closest hit point within a radius d_{max} (or $+\infty$ if no hit occurred)
  - d_{max} can be set to ∞

Ambient Occlusion – Attenuation Function

- $\mu\big(d(\mathbf{p},\omega_i)\big)$ can be any intuitive function
- Simplest case:

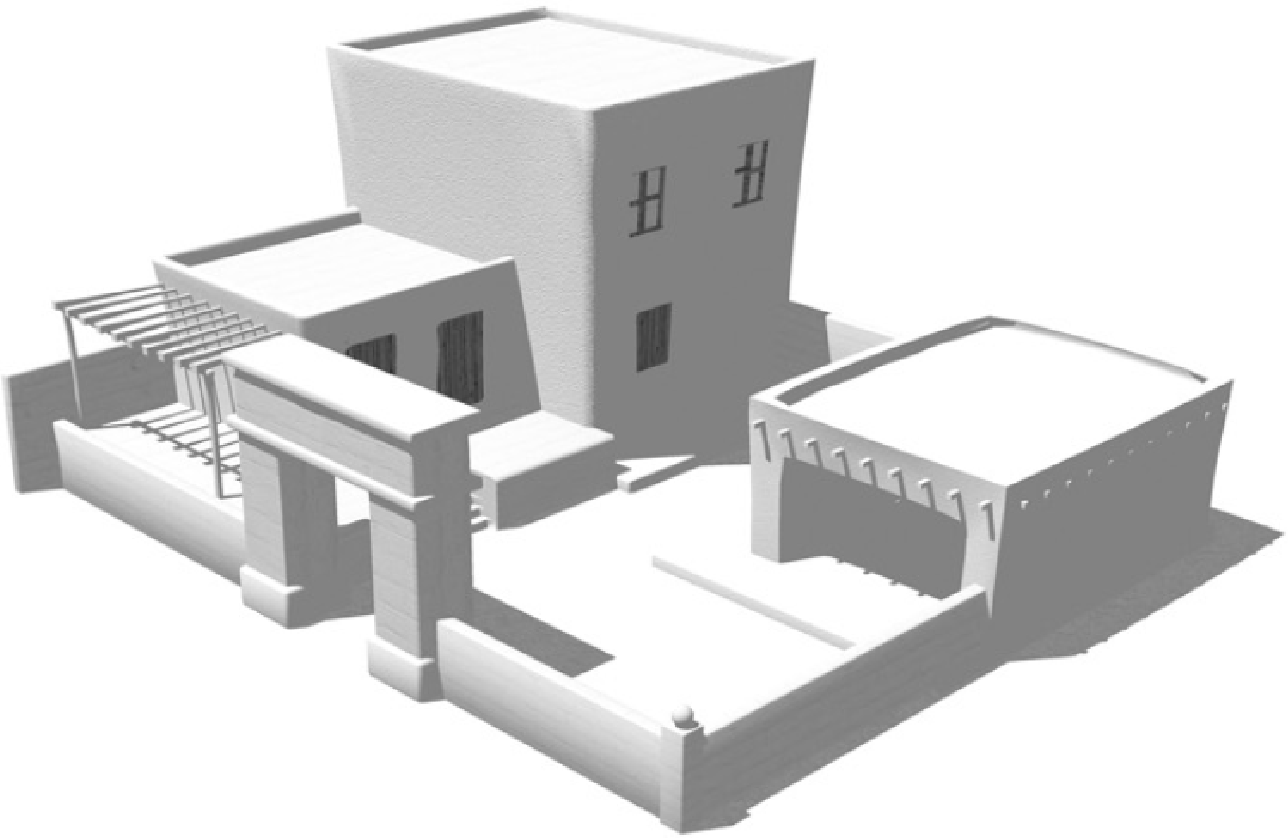
$$\mu\big(d(\mathbf{p},\omega_i)\big) = \begin{cases} 1, & no\ hit \\ 0, & otherwise \end{cases}$$

- But other forms can be used to limit the impact of distant occluders

A.O. : How is it Applied?

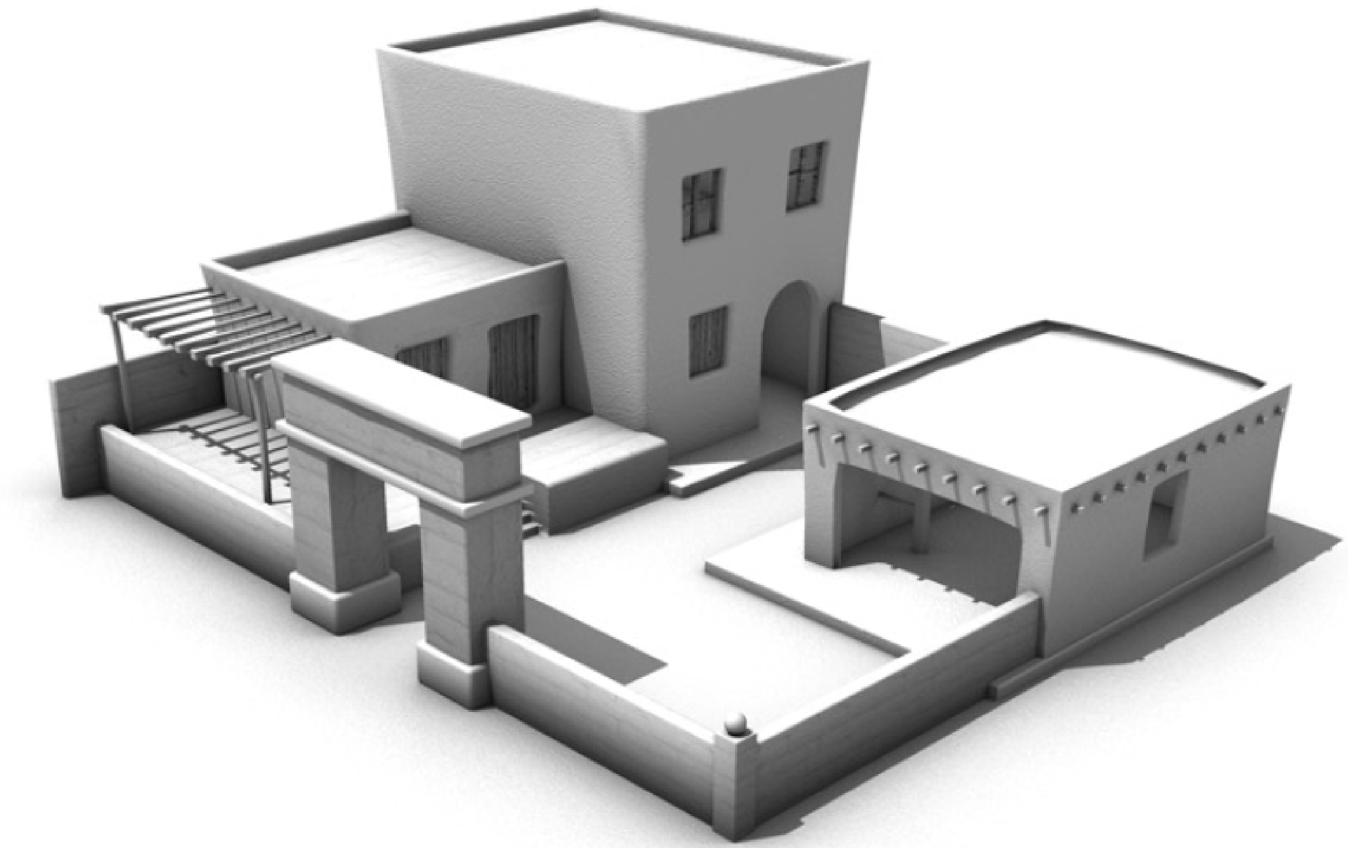
- We usually apply AO as a visibility function to attenuate ambient / sky color
- Some implementations also blend AO with diffuse or even specular lighting (not really correct…)

A.O. Example



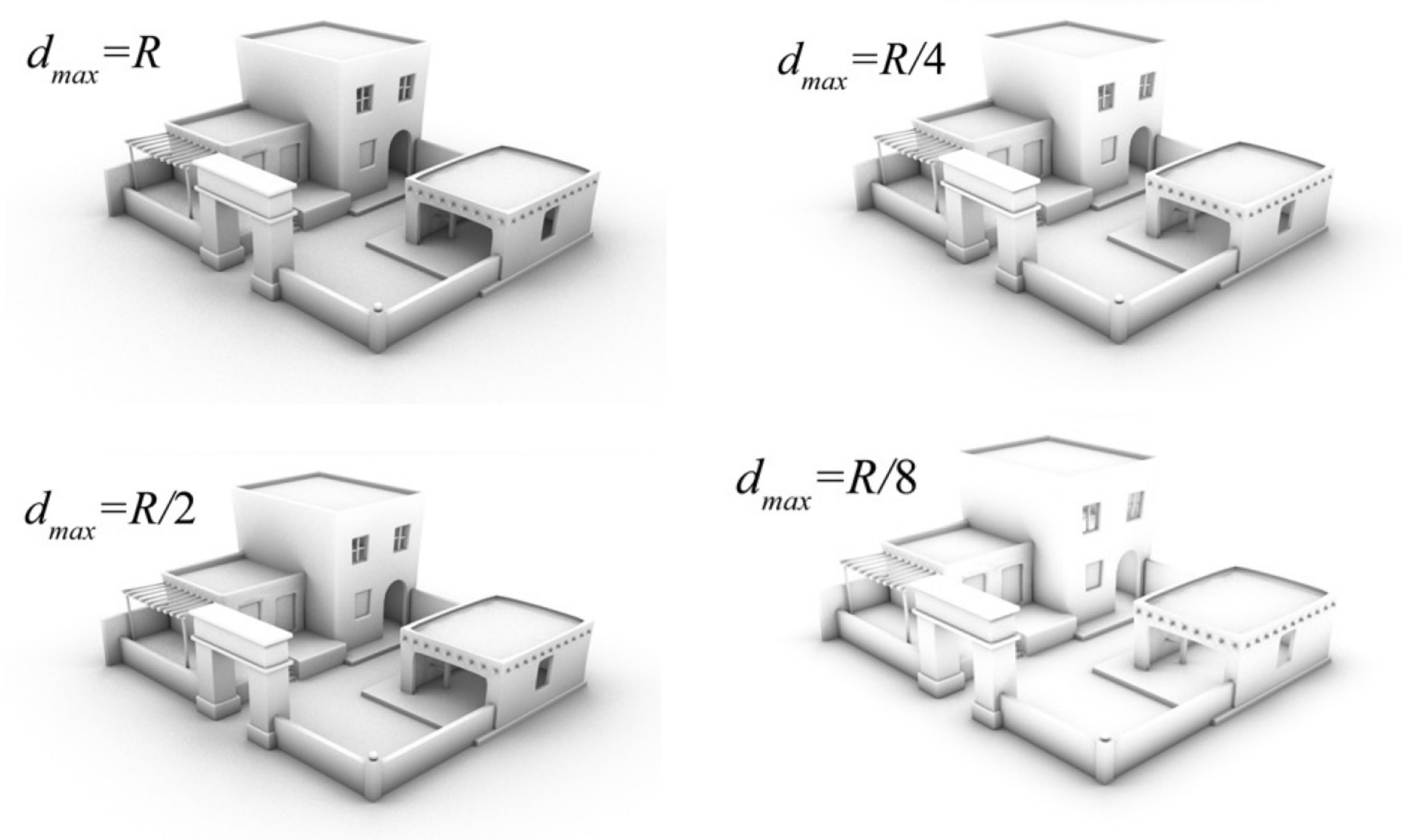
Scene rendered with constant ambient lighting

A.O. Example



Scene rendered with ambient occlusion ($d_{max}=R/8$)

A.O. - Effect of maximum distance

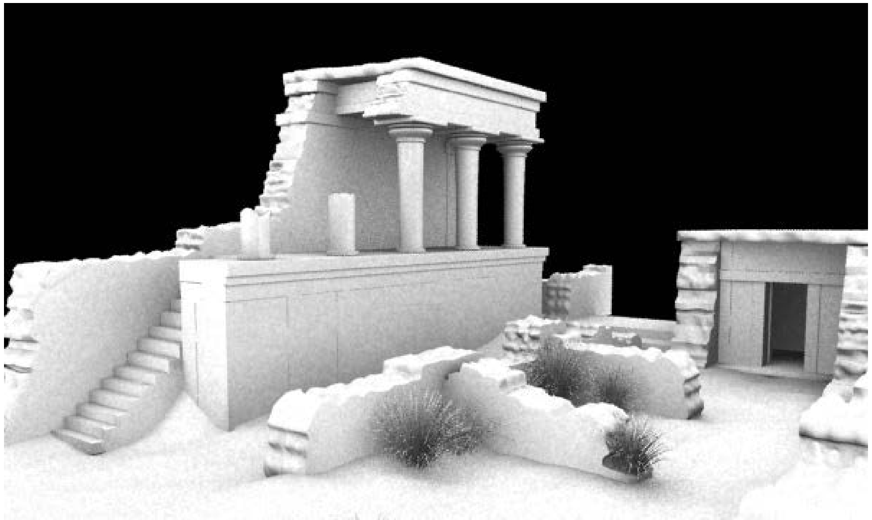


AUEB COMPUTER GRAPHICS GROUP

Ambient Occlusion vs Uniform Light

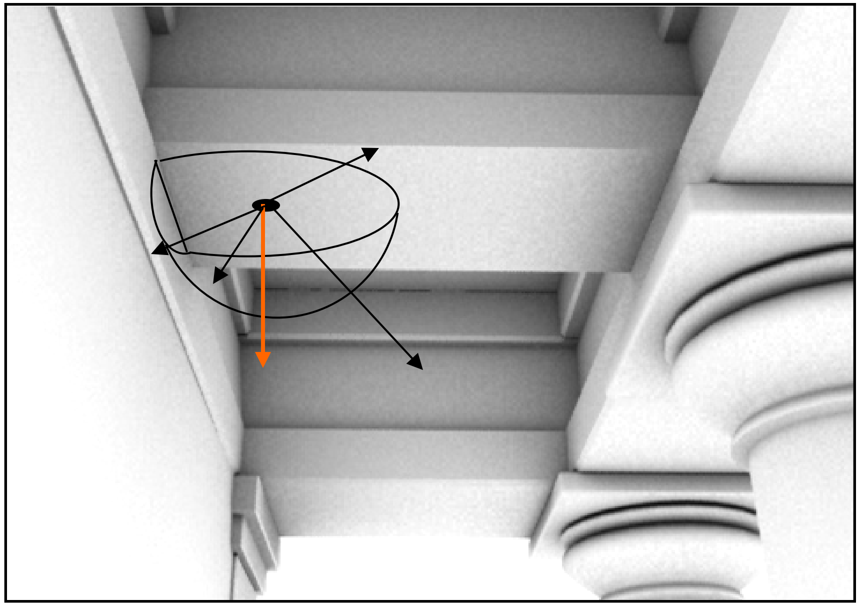
Hemispherical light

Ambient occlusion



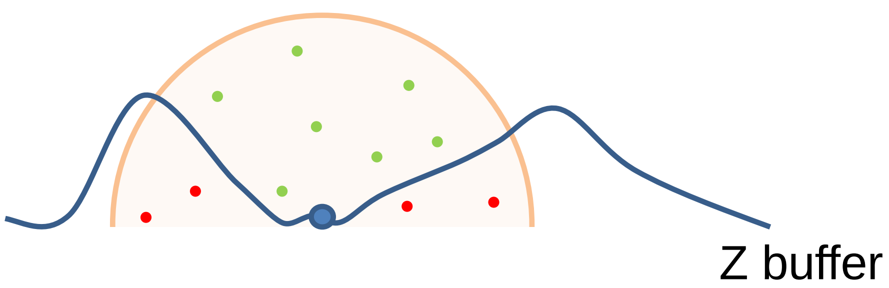
Ambient Occlusion Calculation

- For every visible point $\mathbf{x}$:
  - Compute AO as Monte Carlo hemispherical integral. Sample the hemisphere with N rays:
    - Find closest intersection $\mathbf{y}$ with occluding geometry (the most expensive calculation)
    - Compute distance $d(\mathbf{x},\mathbf{y})$
    - Compute attenuation $\rho(d)$



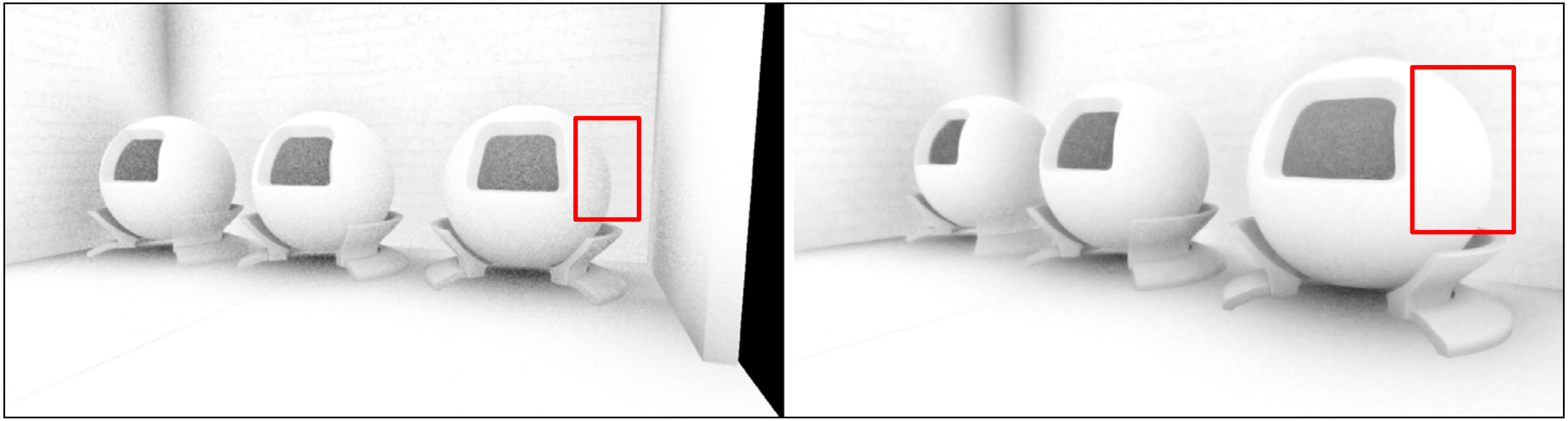
Screen-space Ambient Occlusion

- The most widely used technique for AO in real-time graphics
- Uses the Z buffer as source of occluder geometry information
- Idea:
  - Generate a number of samples up to r_{max} distance away from the shaded point (typically in hemisphere)
  - Test if sample is “above” (in front of) the corresponding z value at that z buffer location
- Many variations



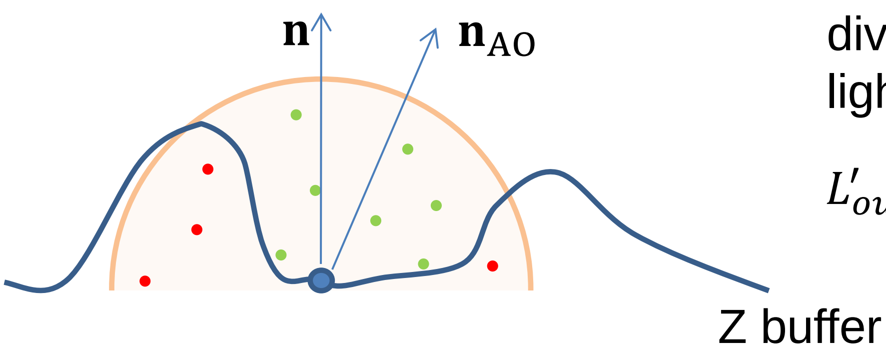
Screen-space Ambient Occlusion

- View-dependent behaviour:
  - Can only use available geometry in view
  - Hidden layers of geometry do not correctly contribute to the result (either over- or under-estimation)



Contact Shadows

- A form of directional ambient occlusion
- Used for attenuating light on surfaces only in directions obscured by nearby geometry
- From the AO samples, compute the average open direction or “bent normal”



Attenuate local illumination by the divergence of the bent normal from the light direction

$$L'_{out} = L_o \mathbf{n}_{\mathrm{AO}} \cdot \mathbf{l}$$

Visibility and Ray Tracing

- Shadows (direct light source visibility) can be also evaluated in real time using ray tracing, on high-end graphics hardware
  - Removes all problematic artifacts of shadow mapping
  - Generally slower

HIGH-DYNAMIC-RANGE RENDERING

Dynamic Range

- Dynamic range: the minimum to maximum luminance level achieved by a system
- The human visual system adapts to the level of illumination incident to the photoreceptors
  - Rods (scotoptic light): $10^{-6}cd/m^2 - 10cd/m^2$
  - Cones (photoptic light): $10^{-2}cd/m^2 - 10^8 cd/m^2$
- Total luminance range: $10^8:10^{-6}$
- Cannot achieve these levels simultaneously!

High Dynamic Range

High Dynamic Range Images - Why

- Physically measured or simulated radiance (therefore luminance) in a natural environment matches the HVS levels
- Typical displays can achieve a **dynamic contrast ratio** of 6000:1 and an **actual luminance level** of 1-120cd/m^2
- Screens are far from capable to display physically correct images!
  - Even if they were, the HVS field of view is different from a screen's → our eyes will not adapt to bright/dark regions appropriately
- We need methods to adapt the computed radiance to the output intensity of a graphics system

High Dynamic Range Images - Storage

- To be able to adjust the tonal range of the image output we need:
  - High precision (float/double) imaging algorithms
  - More than 8bits/color for storage (>255 levels)
  - Floating point precision buffers

- Common settings:
  - RGB16F (48bpp) RGBA16F (64bpp) R11G11B10F - half
  - RGBA12 (48bpp) RGBA16 (64bpp) - int
  - RGB32F (96bpp) RGBA32F (128bpp) - float

Tone Mapping

- Is the process of fitting a potentially huge luminance level to the tonal range of graphics display hardware
- Can be
  - Static
  - Adaptive
  - Delayed adaptive (to simulate the time required for the eyes to adjust to sudden change of illumination levels)
- According to image coverage, it can be
  - Global (same equation and params for all pixels)
  - Local (different adaptation for each pixel)

Tone Mapping - Goals

- De-saturate useful range of information
- Enhance contrast of useful ranges
- Human visual system discriminates changes, not absolute values→
- Local contrast enhancement:
  – Separates tone levels of adjacent pixels →
  – accentuates details
- Simulate the retinal response to physical luminance levels (see blurring and bloom)

Tone Mapping – Maximum to white

- Global operator
- Simple to implement (offline/real-time)
- Assuming normalized output: $L_o = L_i / L_{\max}$
- Ensures mapping of entire range to visible scale
- Reduces contrast for $L_{\max} > 1$
- Increases contrast for $L_{\max} < 1$
- Prone to significantly reduce levels if isolated high values are present

Tone Mapping – Max to white Implementation

- To measure $L_{\max}$:
- Set Blending mode to MAX
- Prepare a 1X1 buffer (single pixel image!)
- Draw the frame
- Read the pixel's value

Tone Mapping – Average Luminance

- In more sophisticated global tone mapping approaches, we evaluate the "general appearance" of an image instead of strict ranges
- We need to evaluate average luminance
- It is preferable to find the log-average of luminance and not the linear one:

$$\bar{L}_w = \exp\left(\frac{1}{N} \sum_{x,y} \log(\delta + L_w(x,y)) \right), \qquad \delta = \text{small float}$$

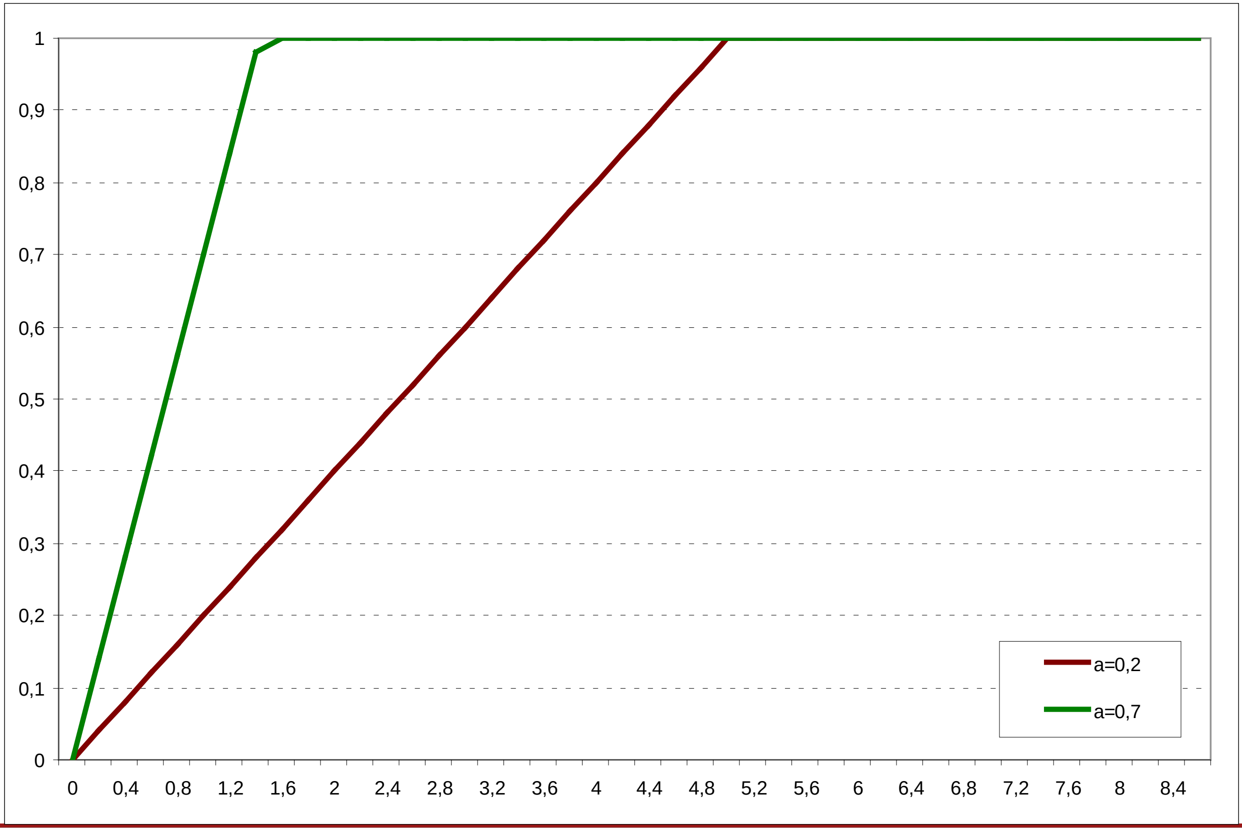
- Because:
  - Perceived intensity on photoreceptors follows the power law
  - So does the working luminance L_w (isolated pixel luminance against a uniform – average – background)

Tone Mapping – Ave. Luminance Implementation

- Goal: measure $\overline{L}_w$:
- Set Blending mode to ADD (normal blending)
- Prepare a small floating point texture as a frame buffer (e.g. 16X16)
- Enable mip-mapping for this texture
- Create a pixel shader to store the log of color as the fragment's resulting color
- Draw the frame
- Read the maximum mip-map level (1X1 texels) and take its exponent. This is the average (estimate over the samples of the low-res buffer)

Tone Mapping – Linear Mapping (1)

$$L_o(x,y) = \min\left\{ \frac{a}{\overline{L}_w} L_w(x,y), L_{o,\max} \right\}$$

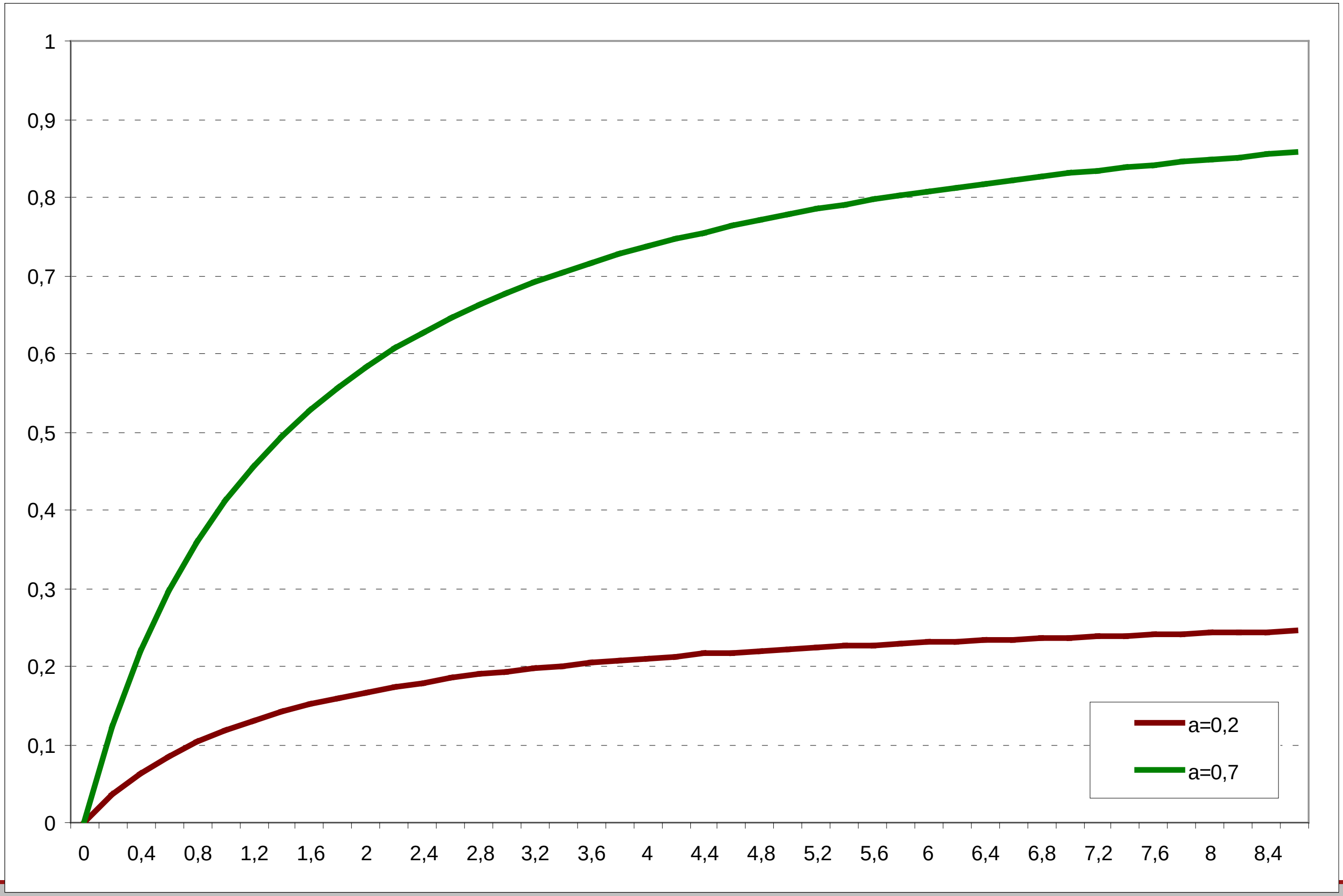


Tone Mapping – Linear Mapping (2)

- a is the tonal "key"
- Clipping
- Global technique
- Easy to implement (off-line/real-time)

Tone Mapping – Non-linear Compression (1)

$$L_d(x,y) = \frac{L_o(x,y)}{1+L_o(x,y)} \qquad L_o(x,y) = \min\left\{\frac{a}{\overline{L}_w} L_w(x,y), L_{o,\max}\right\}$$

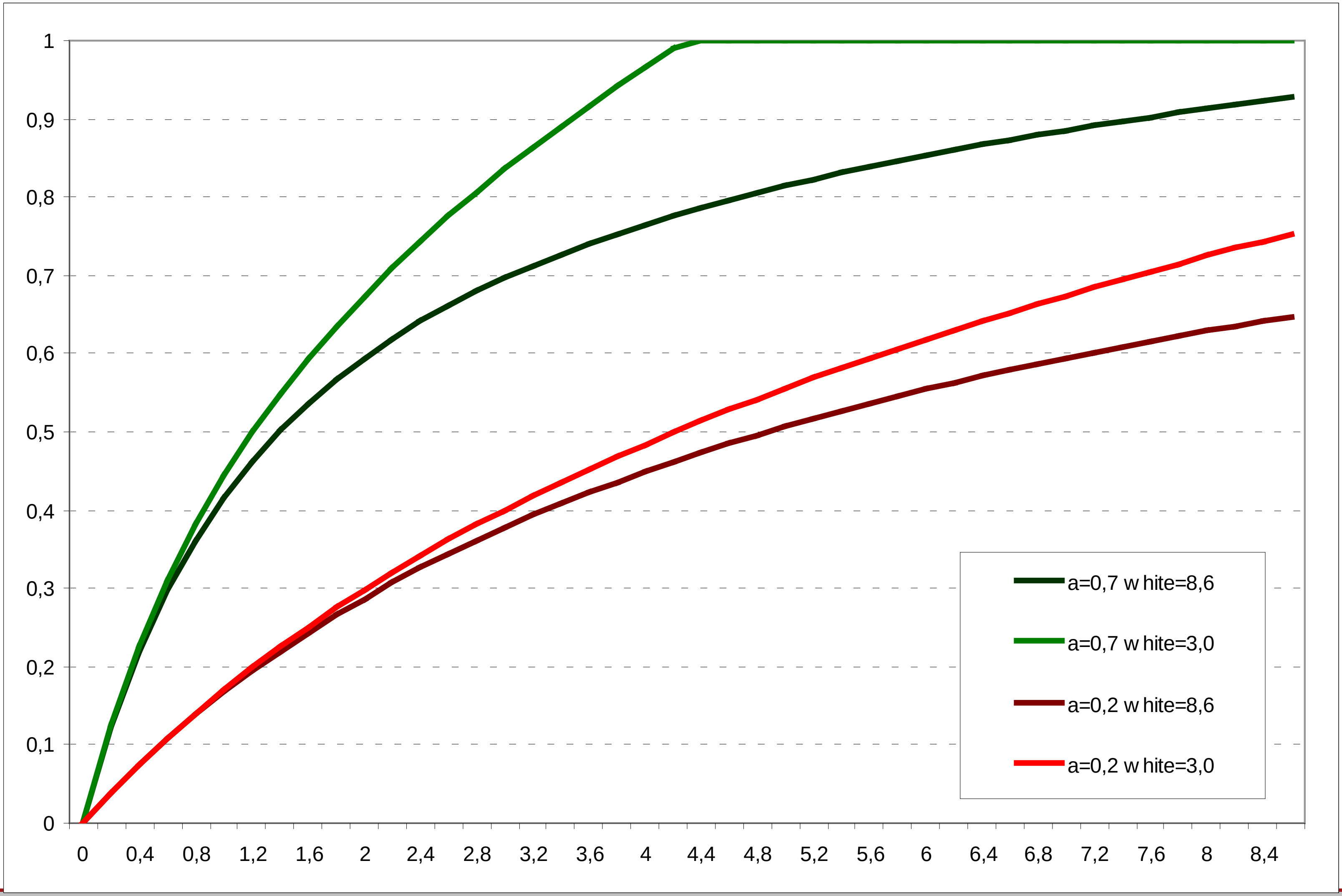


Tone Mapping – Non-linear Compression (2)

- Enhances low-key tonal range
- No clipping
- Better used with a **white point** reference value (expected RGB luminance of “white” – background luminance):

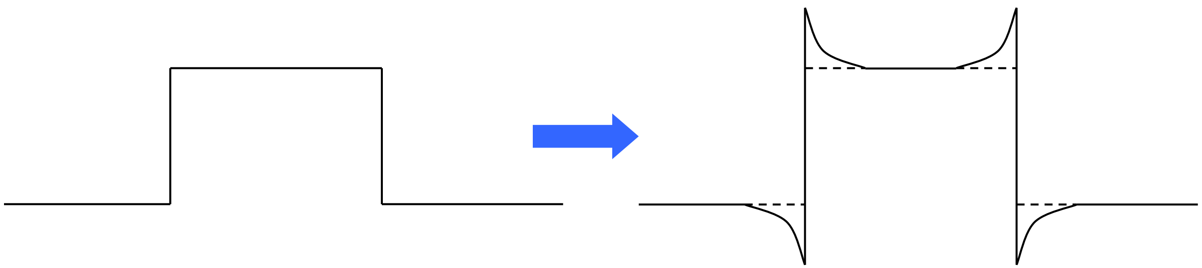
$$L'_d(x,y) = \frac{L_o(x,y)\left(1 + \frac{L_o(x,y)}{L_{white}^2}\right)}{1 + L_o(x,y)}$$

Tone Mapping – Non-linear Compression (3)



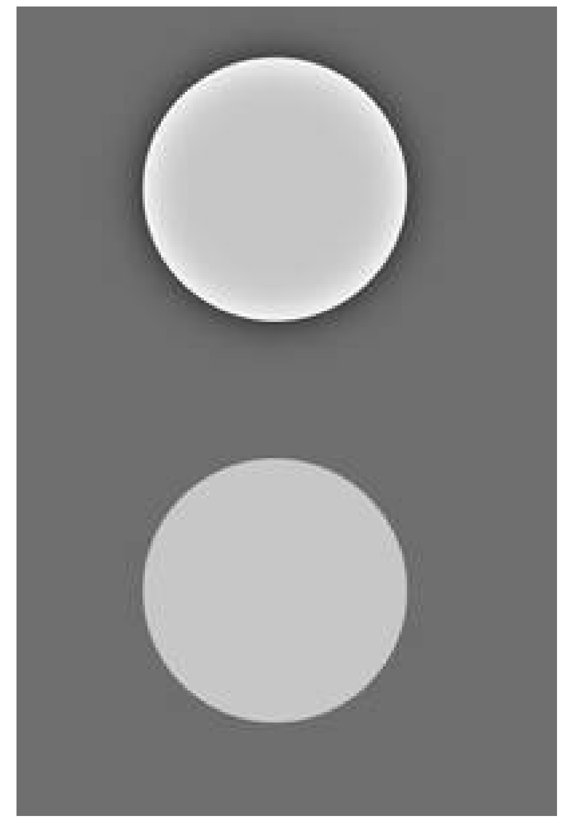
Local Contrast Enhancement

- Local sharpening of the image features gives the illusion of greater dynamic range:



$$I'(x,y) = I(x,y) - s \cdot \nabla^2 I(x,y)$$

$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2}$$



Local Contrast Enhancement Example

SPECIAL EFFECTS

Common In-game Effects

- Bloom
- Motion blurring
- Defocus blurring
- Lens flare

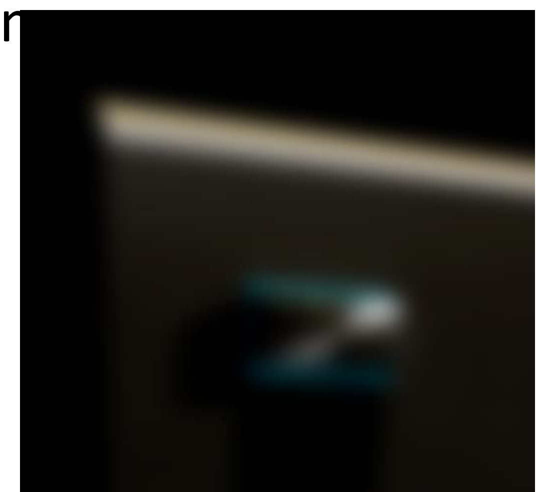
Bloom (1)

- When very bright light is perceived by the human eye, a noticeable glow or intensity “spill” is spread towards the darker regions
- This effect is called bloom and when artificially reproduced in synthetic images, can fool the HVS that an image region is brighter than it really is

Bloom (2)

- To simulate bloom:
  – Subtract a high threshold from the image
  – Blur the result to spread the intensity
  – Modulate the blurred image to achieve the desired effect presence
  – Add to original i

Original



Blurred original-thres

Original+blurred

Real-time Bloom

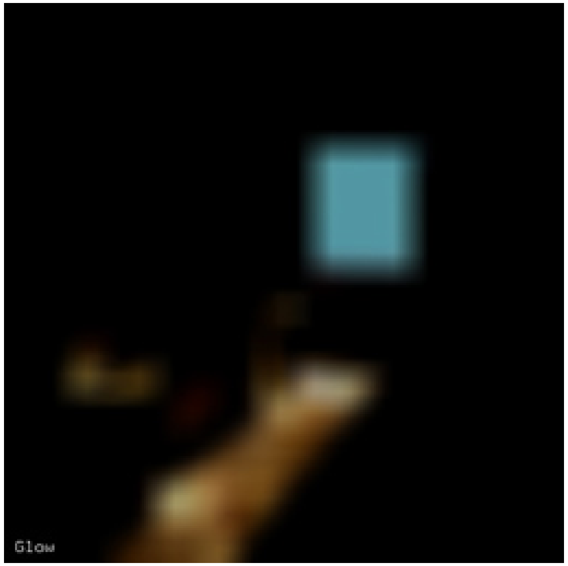
- For real-time rendering bloom is performed similar to off-line rendering
- Blurring (convolution) is an expensive operation
- Requires look-ups and updates over the image → better separate read/store images→ use a “blur buffer”
- Steps:
  - Use a low-resolution frame buffer to store the clipped image
  - Perform upscaling (via bilinear interpolation or/and multisampling) of the low-res buffer
  - Add the result to the image

Real-time Bloom Example

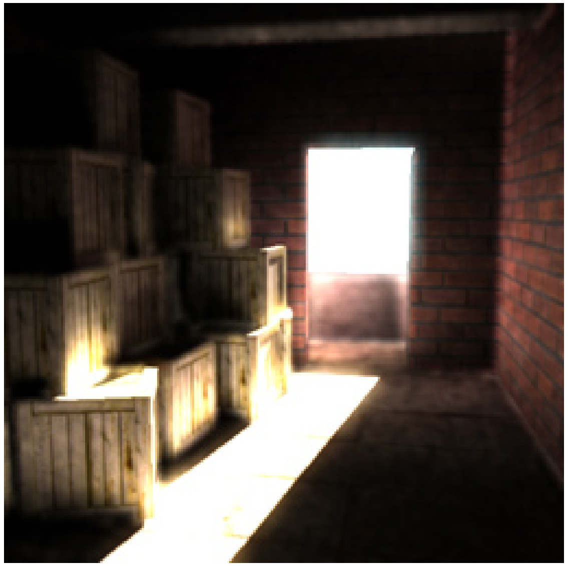
512X512

+

upscaled 64X64

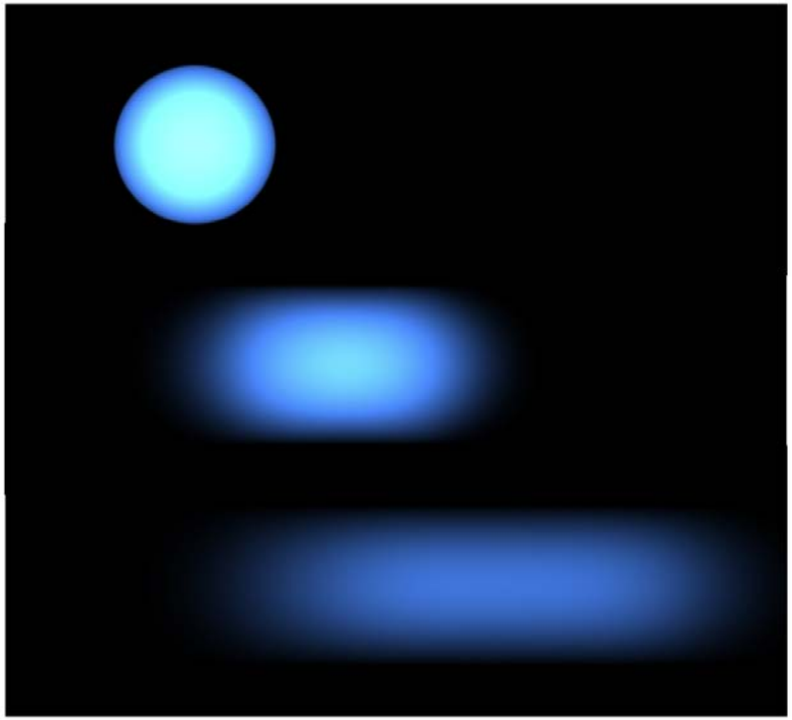


bloom



Motion Blurring

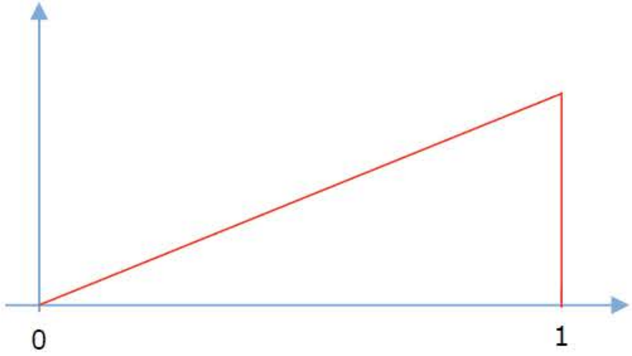
- Given a virtual "shutter", for a fixed exposure time, speed affects the intensity of the resulting image, as energy is "spread" to larger distances:

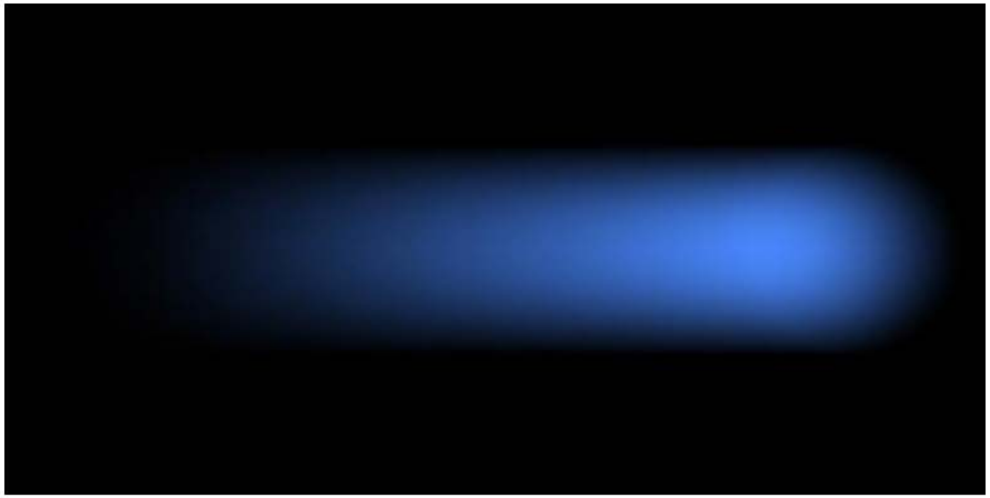


Shutter Profiles (1)

instant open, instant close

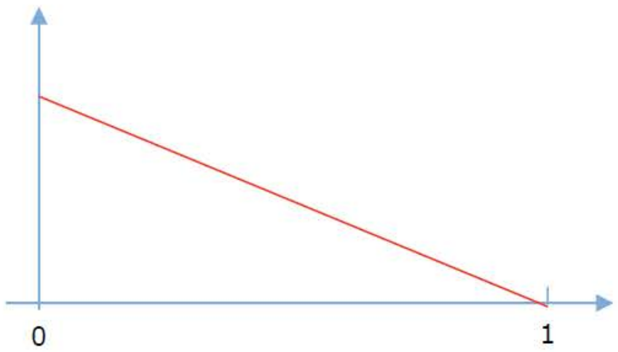
Linear open, instant close

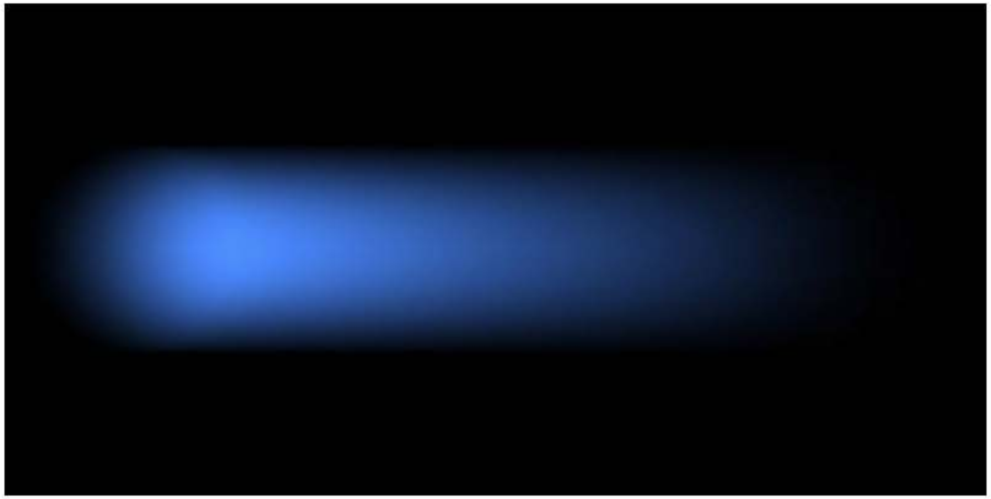




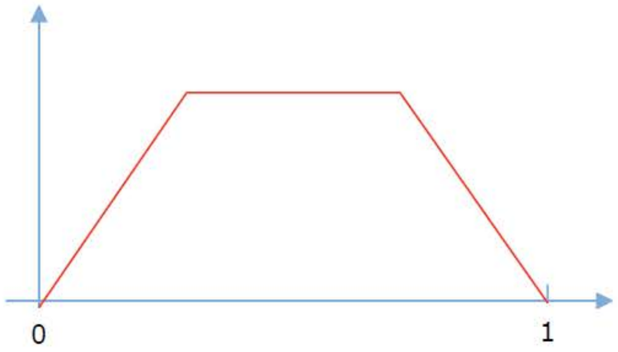
Shutter Profiles (2)

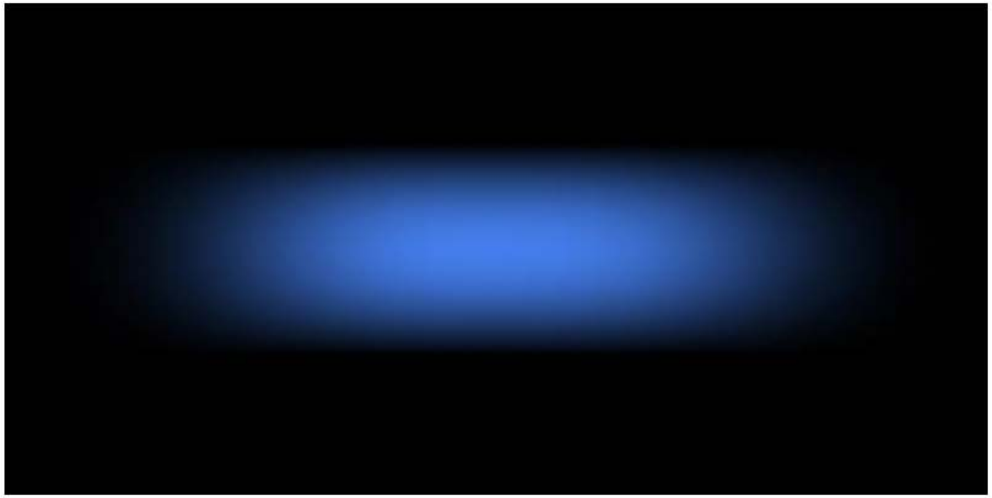
Instant open, linear close





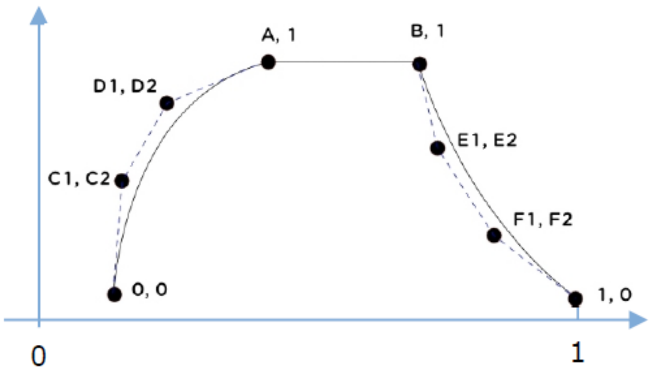
Linear open, linear close

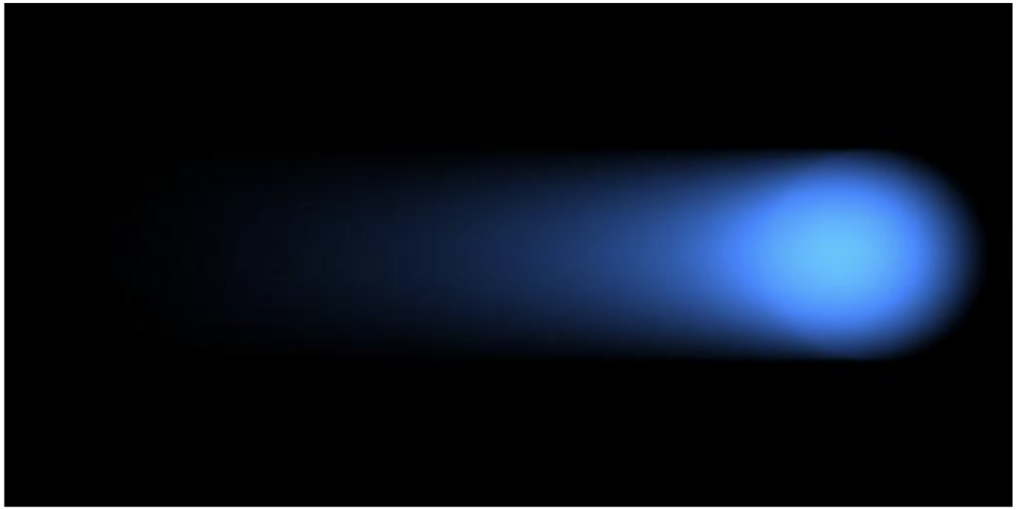




Shutter Profiles (3)

Complex profile

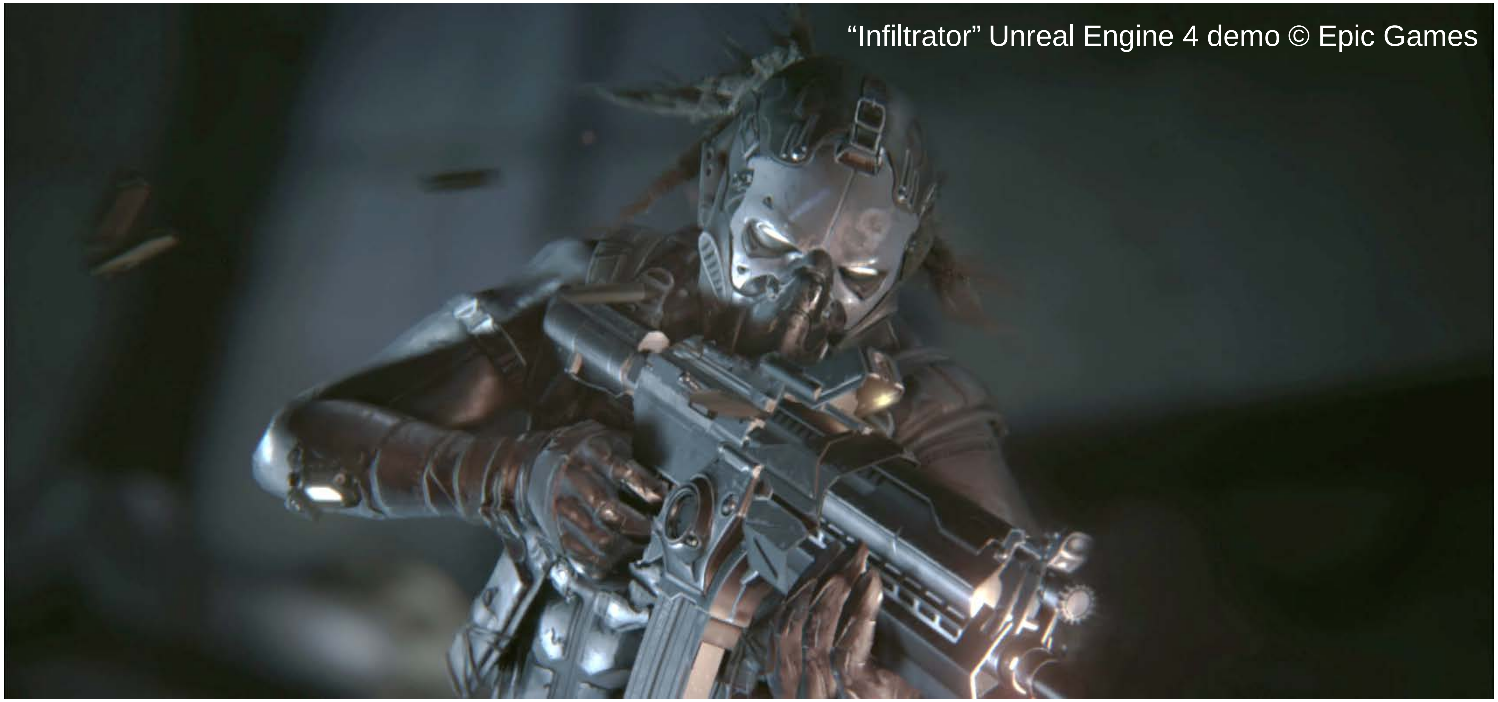




More motion samples towards the end of the shutter interval

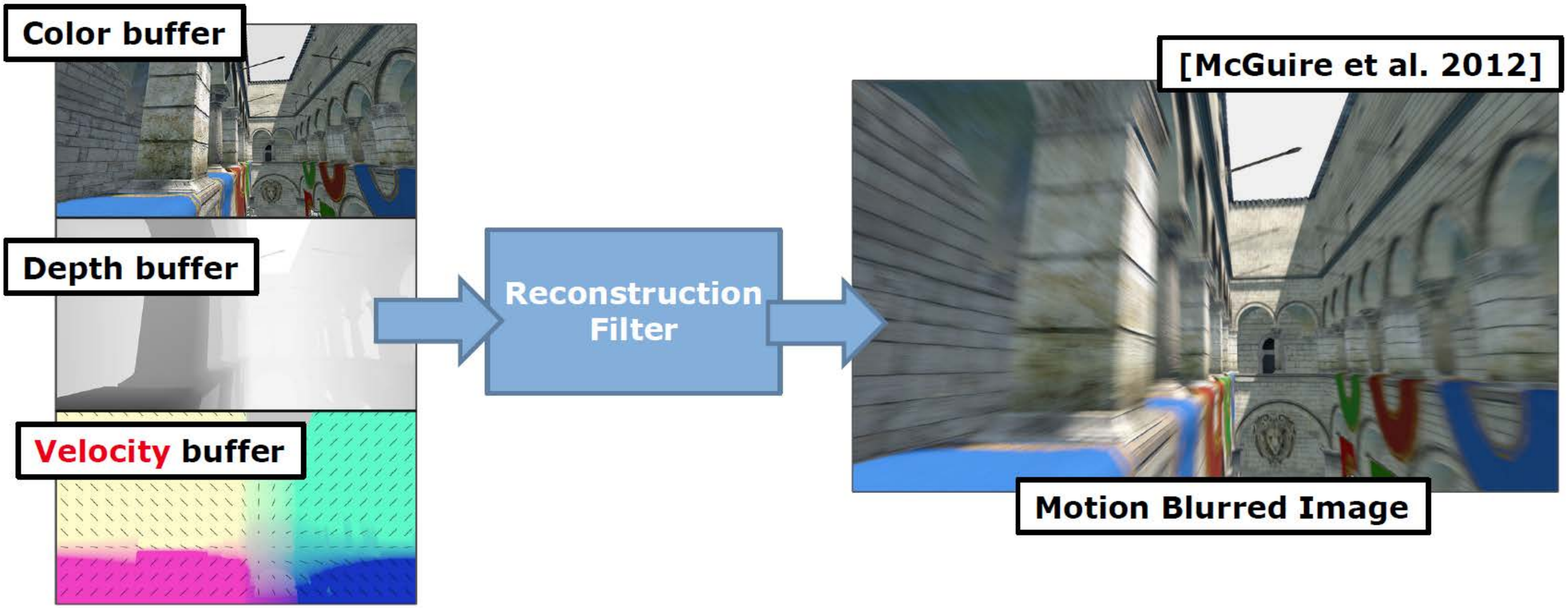
Real-time (RT) Motion Post-filtering

- Re-use samples from previous frames
  - Camera jitter + exponential averaging
  - Motion vectors help recovering fragment position in the past



Motion-blur as Post-process Effect

- Typical solution for **video games** and **real-time** applications



Temporal Pixel Reprojection and Velocity

- Locate the transformed position of the current pixel in the previous frame
  - Retain transformation(s) from the previous frame(s)
  - Transform and interpolate vertices
  - For each pixel obtain transformed positions
  - (optional) store pixel trajectories in velocity buffers

Temporal Pixel Reprojection and Velocity

Depth buffer

Velocity buffer
2 float channels: dx, dy

RT Post-filtering: Re-using Samples

- I found a sample from the previous frame! can I re-use it?
  - Does it come from the right surface?
    - Sample could be from a different object or a mix of objects (e.g. edge → background + foreground)
    - Sample comes from the right object but it has drastically different properties
      - e.g. don't want to re-use samples across the faces of a cube
  - Did the current fragment even exist in the previous frame?
    - Was partially or completely occluded?
    - POV change?
    - Were we even rendering it? (i.e. popped into existence in the current frame)
  - …

RT Post-filtering: Artifacts

Pros:

- Very fast run-time
- Easy to integrate in existing applications

Cons:

- Visibility/occlusion is not properly resolved (can result in artifacts, “incorrect” image)

“A boy and his kite” Unreal Engine 4 demo © Epic Games

Additional Reading

- Moving Frostbite to Physically Based Rendering 3.0 https://seblagarde.files.wordpress.com/2015/07/course_notes_moving_frostbite_to_pbr_v32.pdf
- Real Shading in Unreal Engine 4 https://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf

Contributors

- Georgios Papaioannou