Frame Buffers

Computer Graphics
Instructor: Sungkil Lee

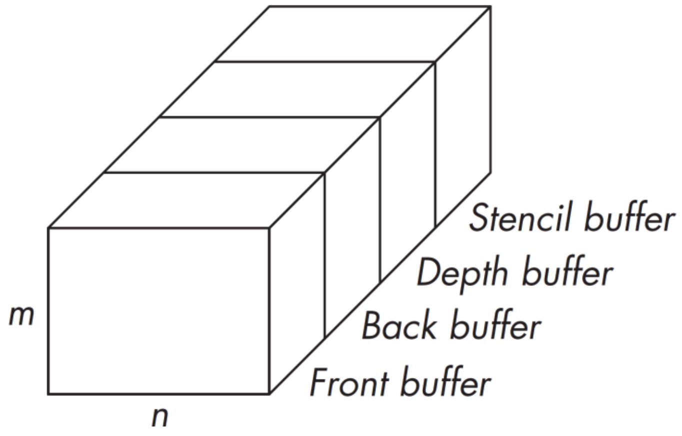
Today

- **Frame buffers**
- **Post-Fragment Operations and Compositing**
  - Depth buffering and blending
  - Compositing
- **Frame Buffer Objects (FBOs) and Render-To-Texture**
  - Frame buffer objects (FBOs)
  - Render-to-Texture
  - Example Applications

Frame Buffers

Frame Buffers

- **Frame buffers store fragments generated via rendering pipeline in large 2D memory areas.**
  - Standard buffers: color buffers and depth buffers
    - Front and back color buffers are defined for double buffering.
  - Additional buffers: stencil buffers, accumulation buffers
    - In many cases, depth-stencil buffers are used together.
  - Stereoscopic rendering uses one more collection (for left and right eyes).



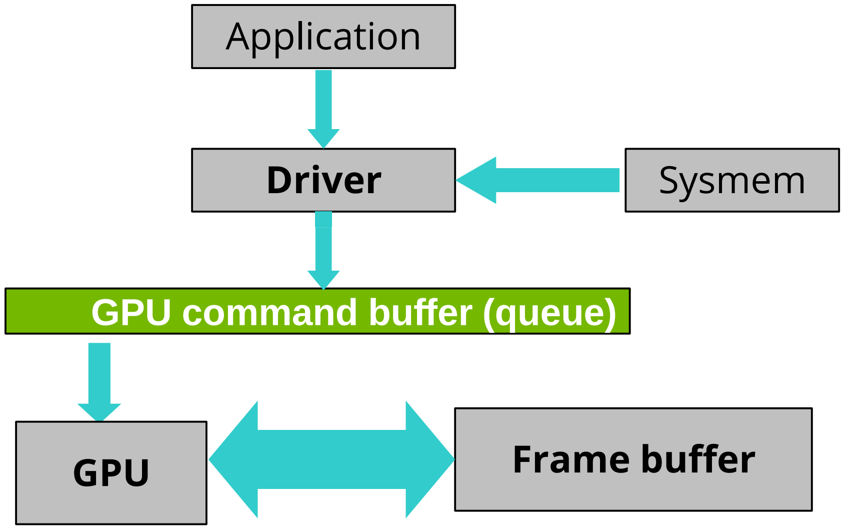
Frame Buffers

- **Frame buffers are defined by resolution and bit depth.**
  - Typical bit depths: 8 bpp, 24 bpp, and 32 bpp
  - This specification is similar to those of textures.
  - Hence, textures can be used as frame buffers, and we call such textures render targets.

- **In a practical scenario, color buffers are solely important.**
  - While color/depth buffers are still actively used, stencil buffers are not that common in modern-style GL.
  - Even reading depth buffers are is nowadays, because we can write our own depth outputs to the color buffers.

Reading from Frame Buffers

- **Read operations**
  - While write operations are simply performed by draw calls by default, read operations can be called but take much longer than write.
  - This results from asynchronous threading model of OpenGL.
  - You can correctly read frame buffers after executing all the pending GPU commands in the GPU command queue.



Flow from application to GPU

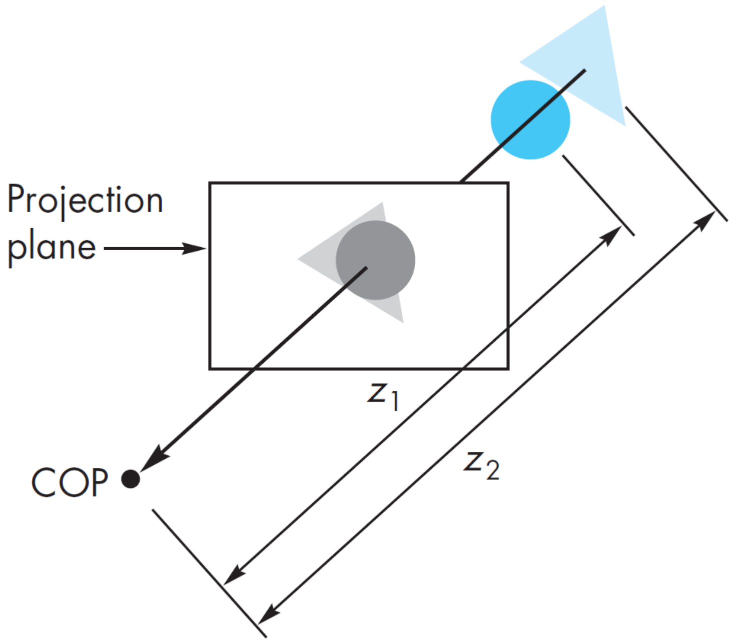
Post-Fragment Operations and Compositing

Post-Fragment Operations

- **Depth test**
  - maintains only nearest fragments in the buffer.
- **Linear blending**
  - alpha blending: use alpha channel
- **Logical operations: bitwise AND, OR, XOR, …**
  - In this case, integer-format frame buffers should be used.
  - Advanced operations such as on-the-fly voxelization often use this.

Depth Buffering

- **Depth test/buffering (Z-test or Z-buffering)**
  - This is the most common operation in raster graphics.
  - When enabling depth test, a fragment's depth is compared against the depth in the frame buffer.
  - When the fragment depth is smaller than the existing depth, the fragment is written to the color buffer with its depth value to the depth buffer. Otherwise, the fragment is discarded.



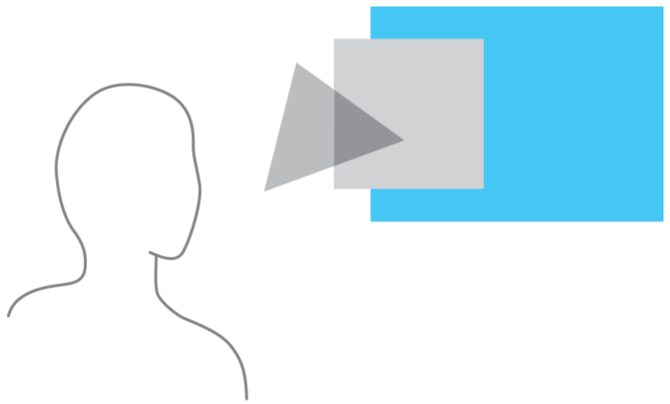
Blending in Frame Buffers

- **Linear alpha blending**
  - Combines the fragment output (with its alpha value) and the colors in the frame buffer (with its alpha value).
  - Source: fragment output values (S_{rgb}, S_a)
  - Destination: framebuffer values (D_{rgb}, D_a)
    - These values are not visible in fragment shaders.

Opacity and Transparency

- **Alpha values (A component in RGBA) now used**
    - "A" means opacity.
    - Translucency = 1 - opacity (or alpha)

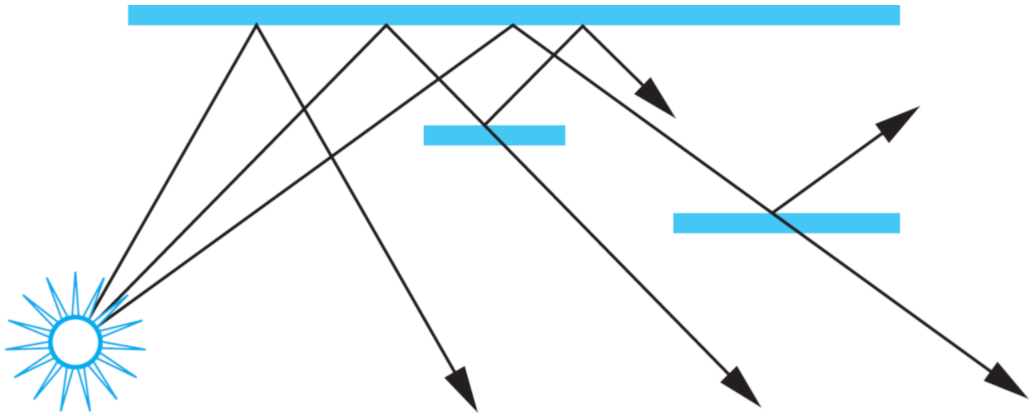
- **General idea on transparency**
    - Opaque surfaces (A=1) permit no light to pass through.
    - Transparent surface (A=0) permit all lights to pass.
    - Translucent surface (0<A<1) permit pass some light.



Translucent and opaque polygons

Translucency: Physical Models

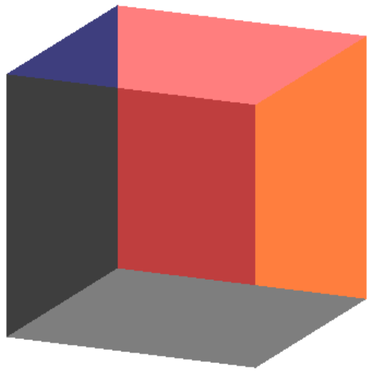
- **Translucency is a challenge in rendering.**
  - Dealing with translucency in a physically correct manner is difficult due to the complexity of the internal interactions of light and matter.
  - In practice, few objects can be approximated.



Scene with translucent objects

Order Dependency

- **Is this image correct? Probably not…**
  - Polygons are rendered in the order they pass down the pipeline
  - However, blending functions are order-dependent.
    - Order-independent transparency is still an on-going research topic.



- **Back-to-front blending**
  - Translucent objects are sorted, and then, are blended (or rendered) in a back-to-front order.

Blending for Transparency

- **Simplest alpha blending**

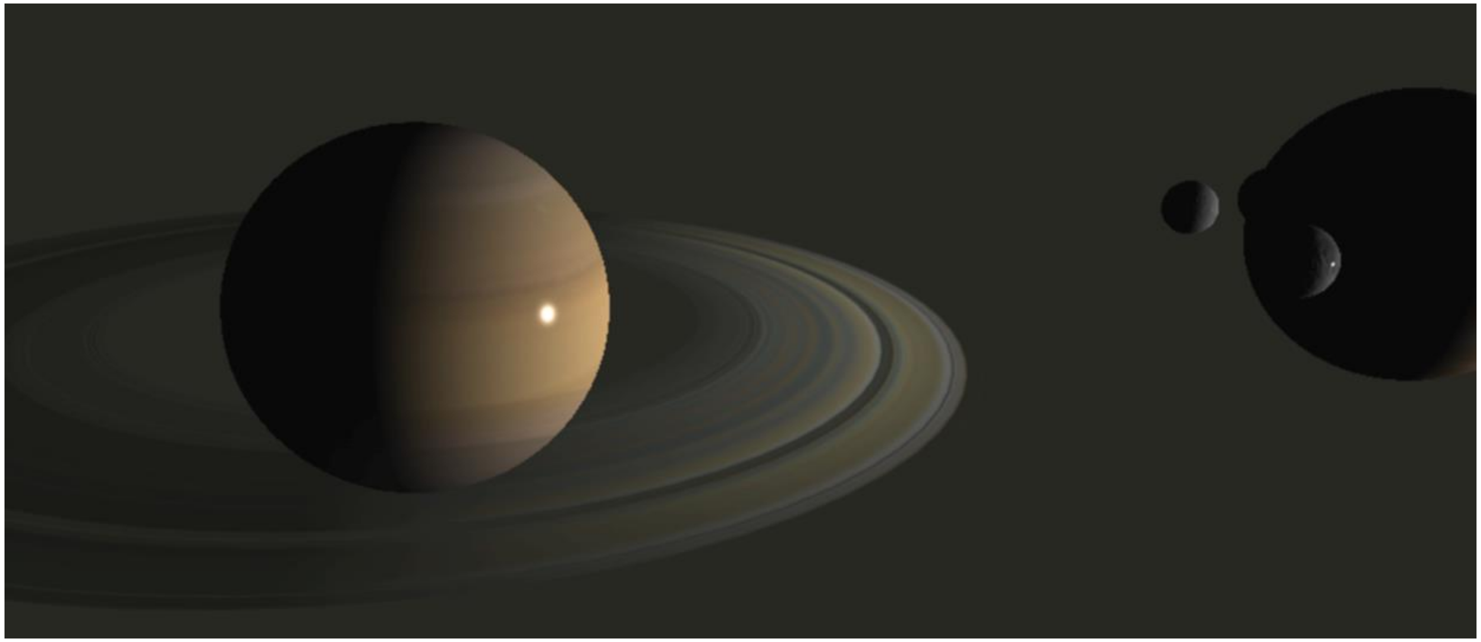
$$D'_{rgb} = S_{rgb} * Sa + D_{rgb} * (1 - Sa)$$
$$D'_{a} = Sa * Sa + Da * (1 - Sa)$$

- **OpenGL example**

```
glEnable( GL_BLEND );
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
...
```

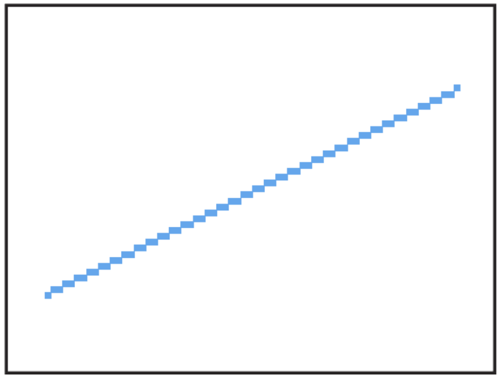
Blending for Transparency

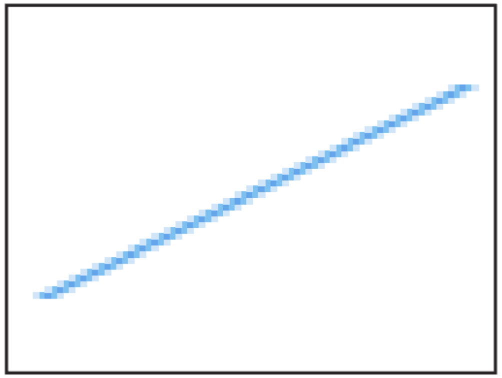
- **Example: the ring system of the Saturn**
  - Draw opaque objects first.
  - Then, blend/render translucent rings in a back-to-front order.



Blending for Transparency

- **Another example with antialiasing**
  - Point/line/polygon are smoothed for antialiasing.
  - The coverage factors to neighboring pixels are computed, and alpha-blended.


original jaggy line


anti-aliased line

  - OpenGL example:

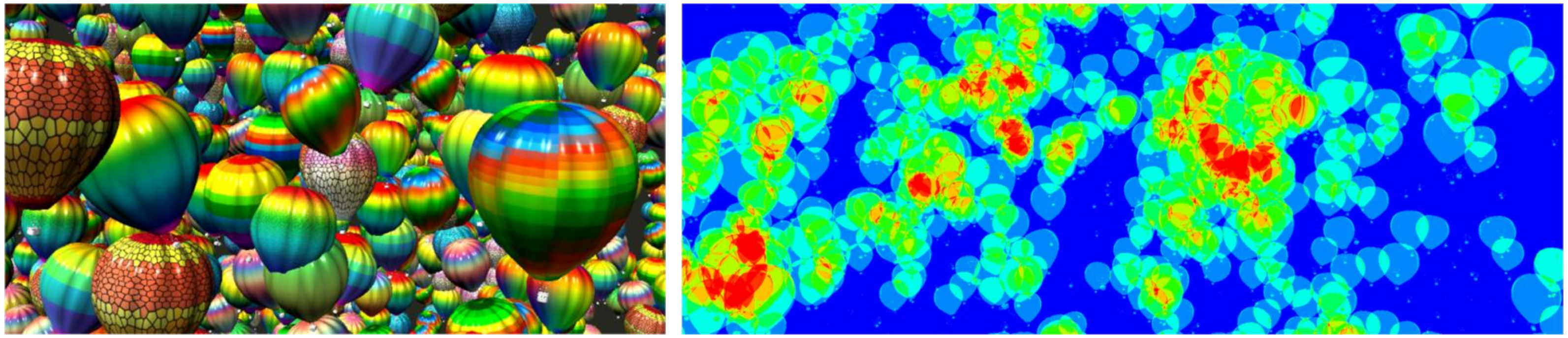
```
glEnable( GL_LINE_SMOOTH );
glEnable( GL_BLEND );
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
```

Additive Blending

- **Additive/accumulative blending**

$$D'_{rgb} = S_{rgb} * Sa + D_{rgb} * 1$$
$$D'_{a} = S_{a} * Sa + Da * 1$$

- **Example**
  - Counting of the number of drawn fragments


Color-mapped number of incoming fragments

Compositing

- **Compositing**
  - When you are blending pre-rendered textures, you can easily implement blending in your shaders.
  - This is called the compositing rather than blending.
  - Compositing is common in many UI/web rendering engines.

- **Examples of the compositing**
  - You have background textures.
  - You have foreground objects with alpha (e.g., leaves of a tree)
  - You can composite the background and foreground objects using alpha blending equations, while disabling blending.

Frame Buffer Objects (FBOs)

Frame Buffer Objects (FBOs)

- **Textures can be used as frame buffers,**
  - because they reside inside the physically same memory areas.
- **We perform the same renderings, yet with our own frame buffer (objects).**
  - We need to specify additional frame buffers whose render targets are attached to textures to replace the default frame buffers.
  - Such frame buffers are called the frame buffer objects.

Render-to-Texture (RTT)

- **Frame buffer objects allow us to perform "render-to-texture."**
  - The rendering results are now stored in the texture memory.
  - We may read them back into main memory for further uses.
  - We may use them as if a static texture in shaders.
  - RTT is a primitive for many postprocessing.

- **Example: defocus blur**
  - Pass 1: render the objects' RGB colors and depths
  - Pass 2: apply a box blur by the distance to the focal plane depth.

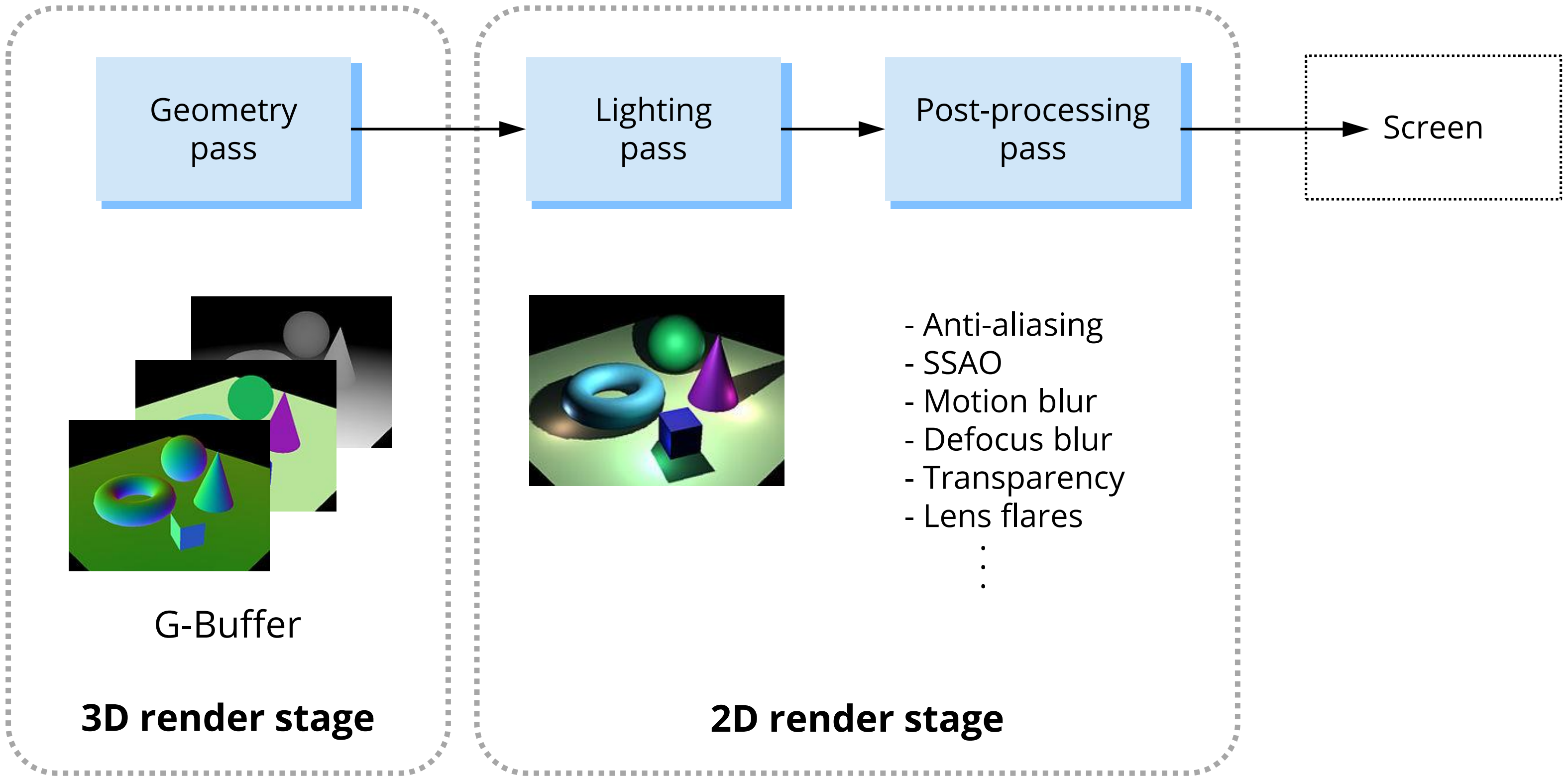
Render-to-Texture: Example Applications

Deferred Rendering

- **General idea**
  - Multi-pass approach yet with a single geometry pass
  - Render attributes of the geometry and its material first on geometry pass.
    - The buffer is called the *G-buffer*.
    - Common attributes: position, normal, diffuse color, specular
  - Perform shading later after the geometry pass
  - Apply postprocessing to the lit image with additional attributes
    - Depth, motion vectors are common

Deferred Rendering

- **Overview**

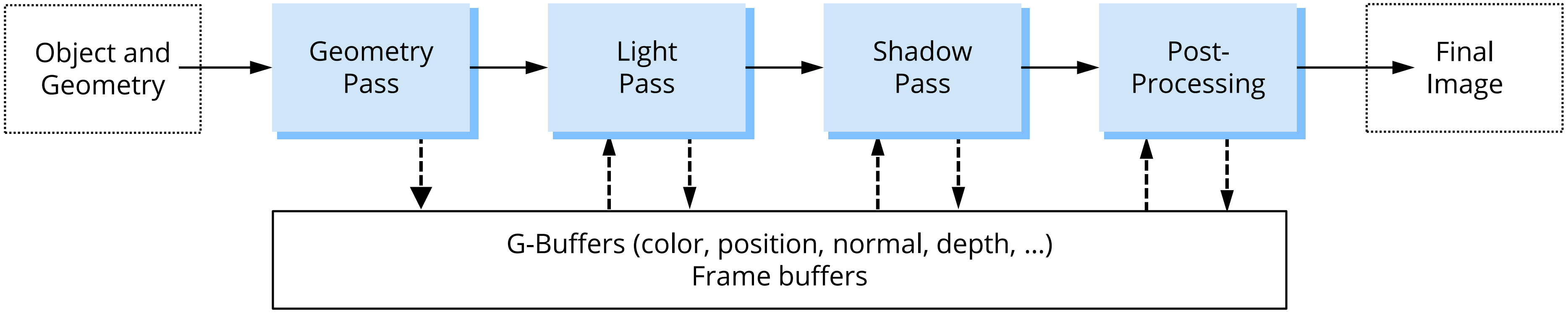


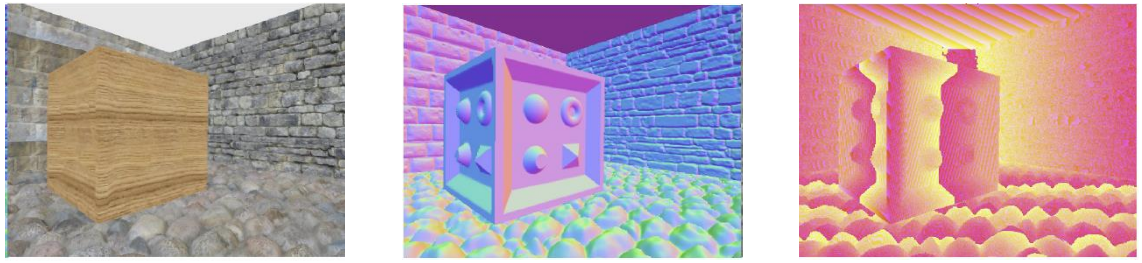
Deferred Rendering

- **Commercial engines using deferred shading**
  - Nearly all the major game engines

Geometry Pass

- **Attributes to render to G-buffer in geometry pass**
  - Common four attributes:
    - Normal, depth, diffuse color (albedo), specular color/shininess





[Thaler 2012]

Geometry Pass

- **Multiple render targets (MRTs) for G-buffers**
  - G-buffers storing all the attributes need more than the maximum capacity of a single buffer (typically 128 bits, 4 channels)
  - MRTs allow us to simultaneously render the attributes in a single pass
    - Up to 8 back buffers in general.
  - Nonetheless, the amount of channels need to be reduced.
    - Bitwise encoding/packing are required
    - 4 channels $\times$ 8 bits = 32 bits / render target

Geometry Pass

- **Example of the packed data structure for G-buffers**

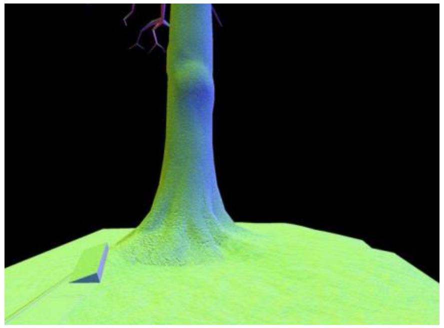
	R8	G8	B8	A8
DS	Depth 24bpp			Stencil
RT0	Lighting accumulation RGB8			Intensity
RT1	Normal.x FP16		Normal.y FP16	
RT2	Motion vectors XY RG8		Specular.power	Specular.intensity
RT3	Diffuse albedo RGB8			Sun.occlusion

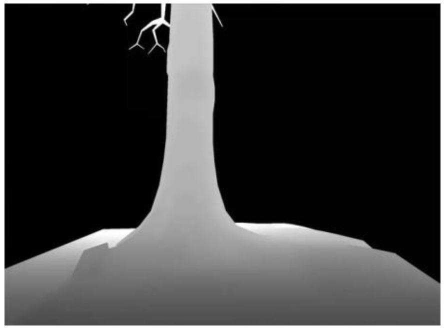
The Rendering Technology of Killzone 2

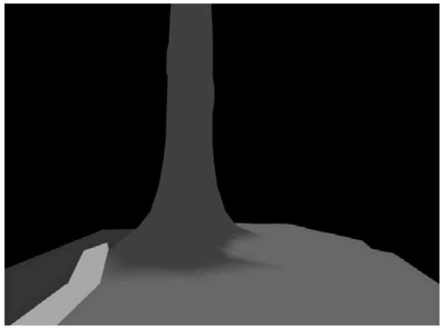
- Z of normal vector can be calculated from X and Y.
- Limitations: low precisions and overhead of packing/unpacking
- Yet, cheaper than bandwidth overhead

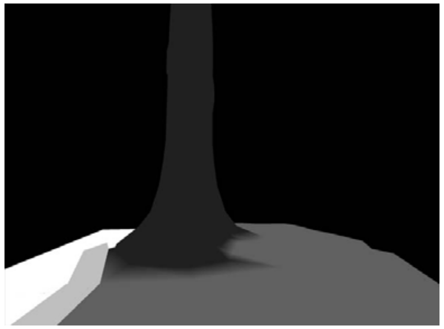
Examples

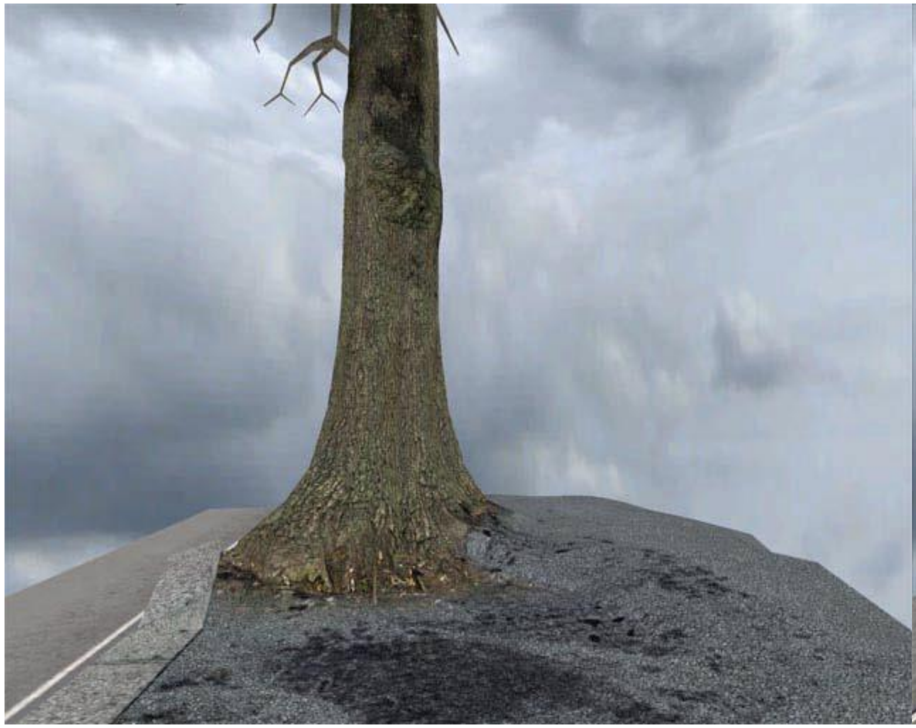
Deferred Shading, Shawn Hargreaves, GDC 2004

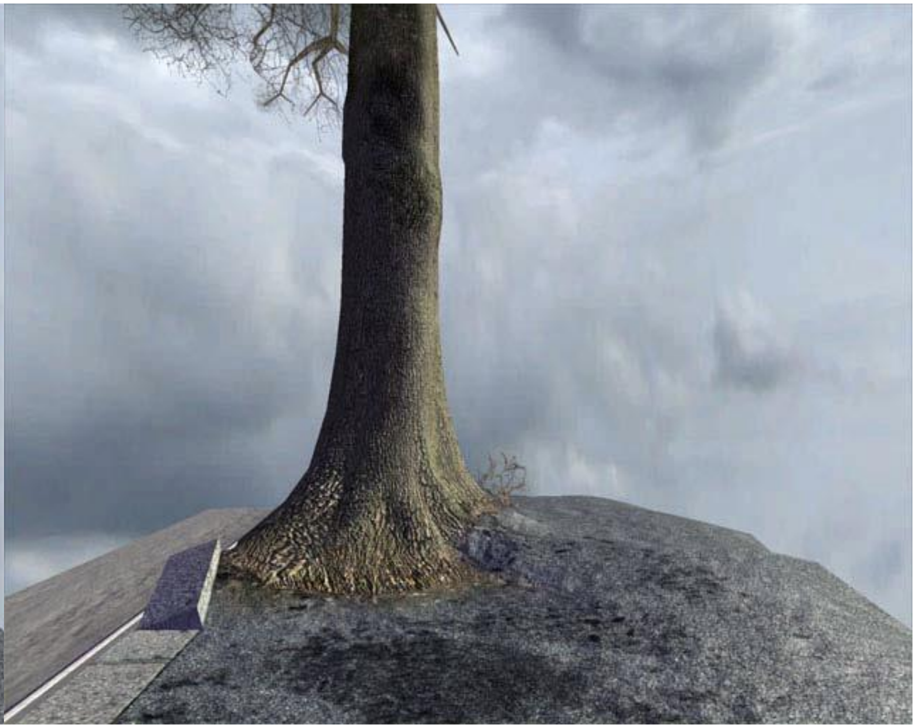

normals


depth


specular intensity

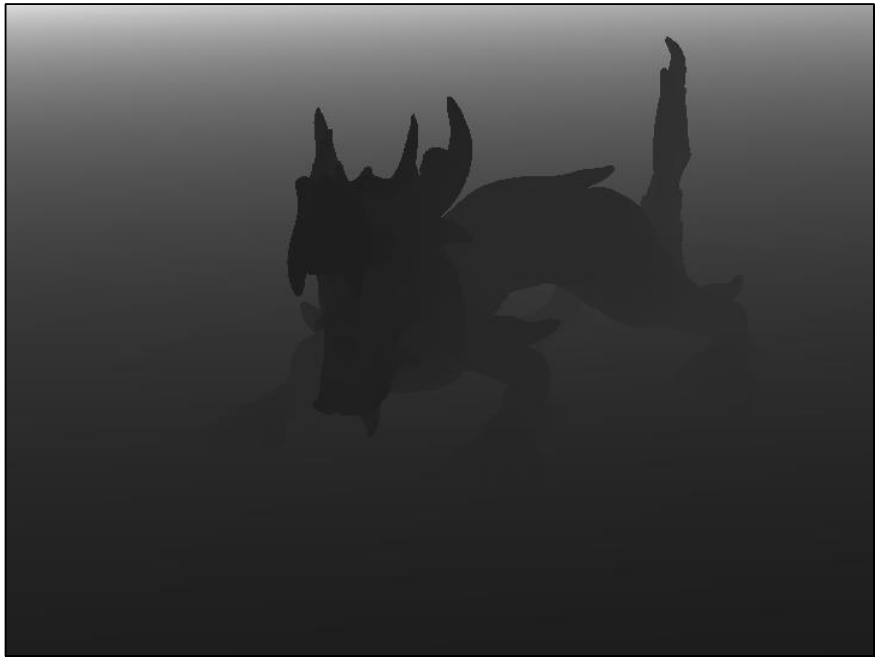

specular power

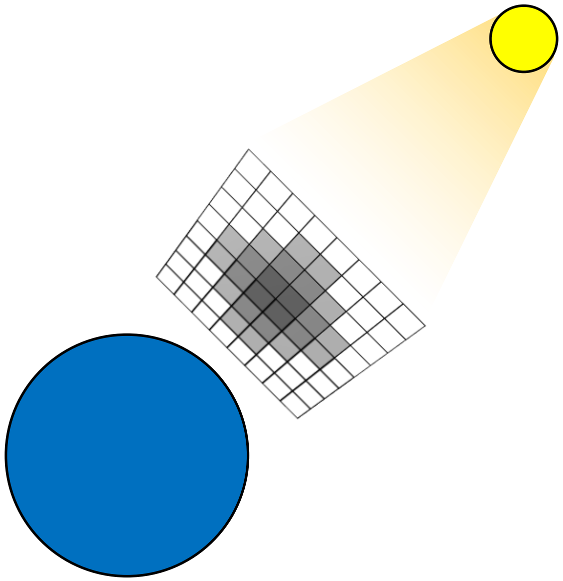

Diffuse color (albedo)


Deferred lighting result

Shadow Mapping

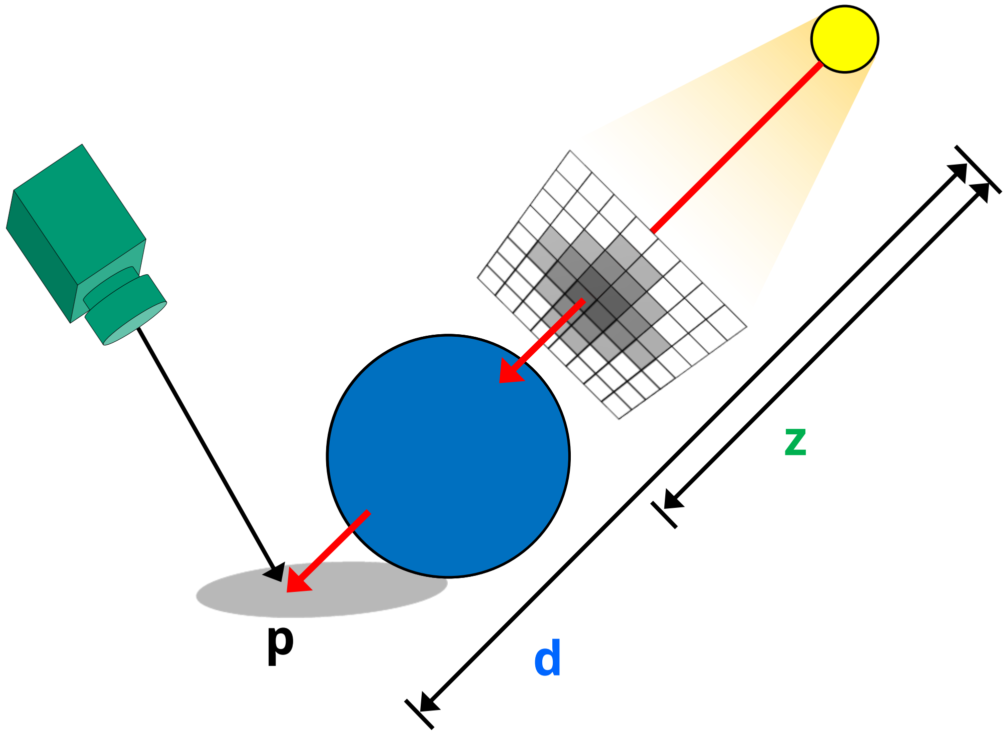
- **First pass: shadow map rendering**
  - Render the depth image (shadow map) from the light viewpoint.





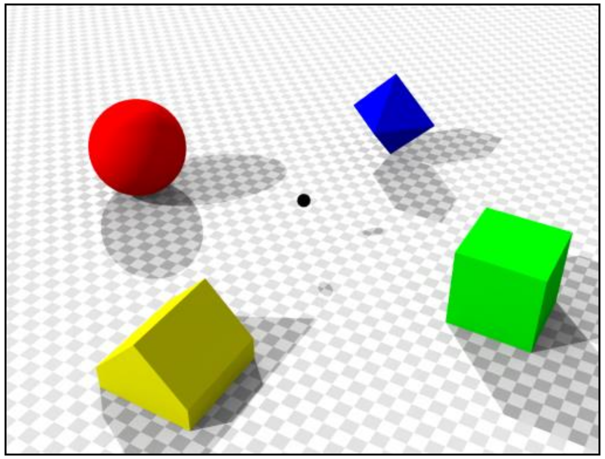
Shadow Mapping

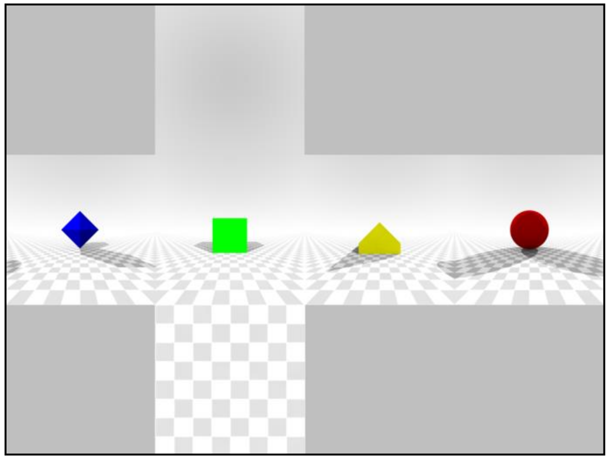
- **Second pass: calculating shadow factors**
  - Calculate shadow factors for each fragment that determines whether the fragment is shaded.
    - Compare depth values of the current fragment (z) and the shadow map (d).
    - Point p is lit (no shadows) if d>=z.
    - Point p is occluded if d<z.

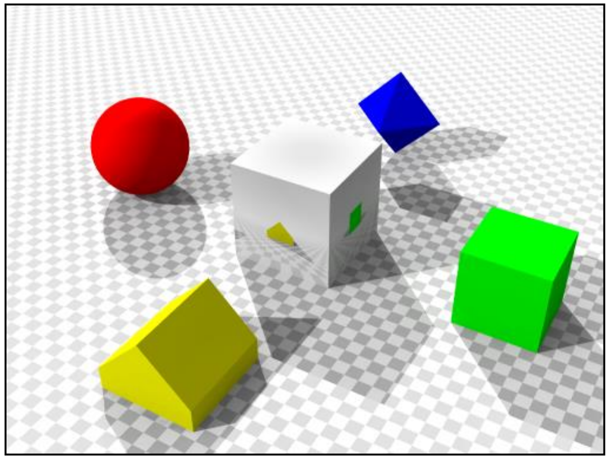


Online Cube-Map Rendering

- **Cube mapping:**
  - Environment mapping via a 6-face cube map
  - Could be a static texture or dynamically-generated for every frame


place a viewer in a scene


render 6-face cube map


apply texture to the surface