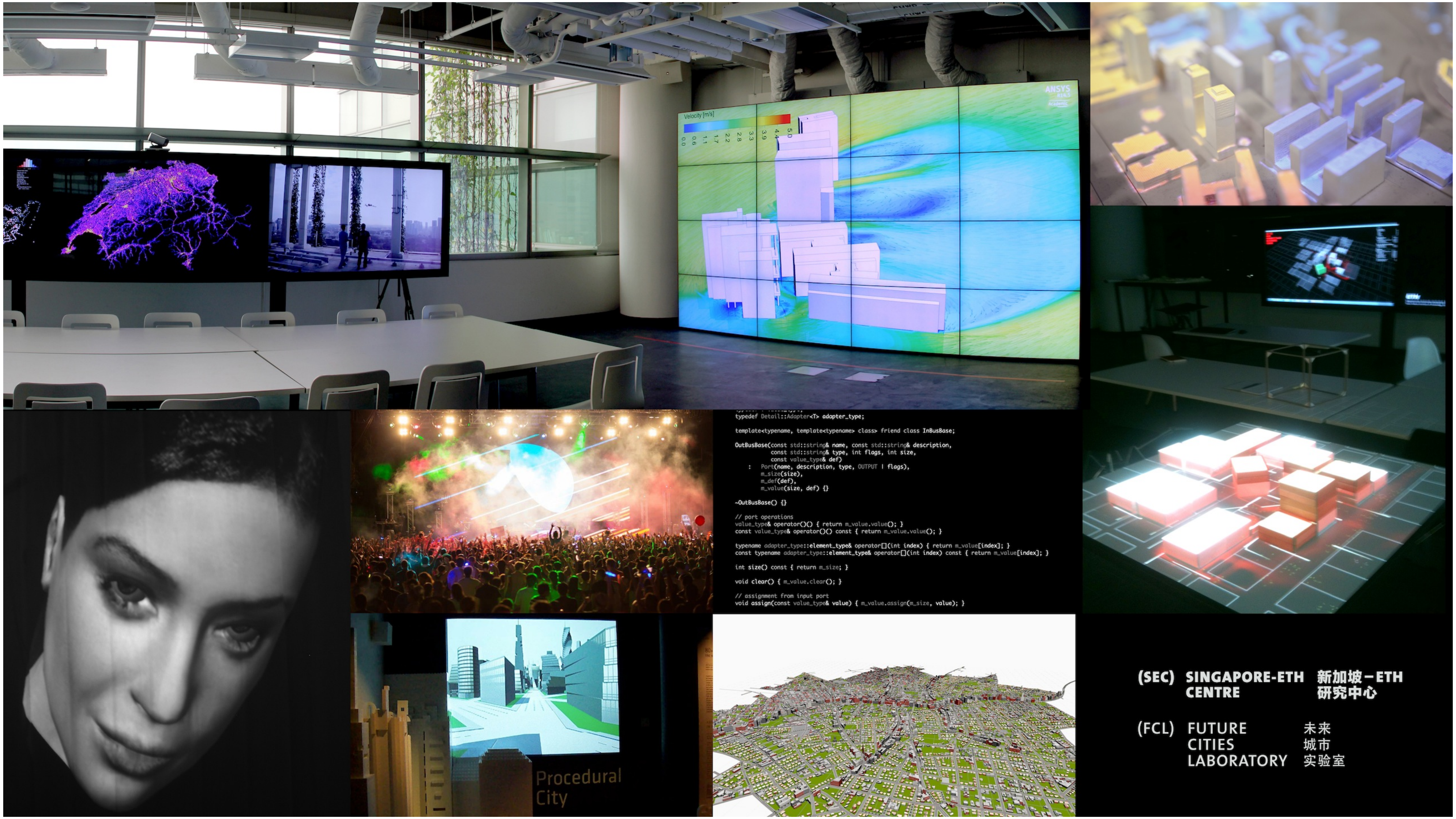
DEFERRED RENDERING

STEFAN MÜLLER ARISONA, ETH ZURICH

SMA/2013-11-04


typedef Detail::Adapter<T> adapter_type;
template<typename, template<typename> class> friend class InBusBase;
OutBusBase(const std::string& name, const std::string& description,
const std::string& type, int flags, int size,
const value_type& def)
: Port(name, description, type, OUTPUT | flags),
m_size(size),
m_def(def),
m_value(size, def) {}
~OutBusBase() {}
// port operations
value_type& operator()() { return m_value.value(); }
const value_type& operator()() const { return m_value.value(); }
typename adapter_type::element_type& operator[](int index) { return m_value[index]; }
const typename adapter_type::element_type& operator[](int index) const { return m_value[index]; }
int size() const { return m_size; }
void clear() { m_value.clear(); }
// assignment from input port
void assign(const value_type& value) { m_value.assign(m_size, value); }
Procedural
City
(SEC) SINGAPORE-ETH CENTRE
新加坡－ETH 研究中心
(FCL) FUTURE CITIES LABORATORY
未来 城市 实验室

DEFERRED RENDERING?

CONTENTS

LEARNING GOALS

- Understand the motivation for DR and how it is different from forward rendering.
- Know the advantages, challenges, and limitations of DR.
- Understand example uses of DR (shading, AO).
- Know the basic mechanisms in OpenGL to realize DR.

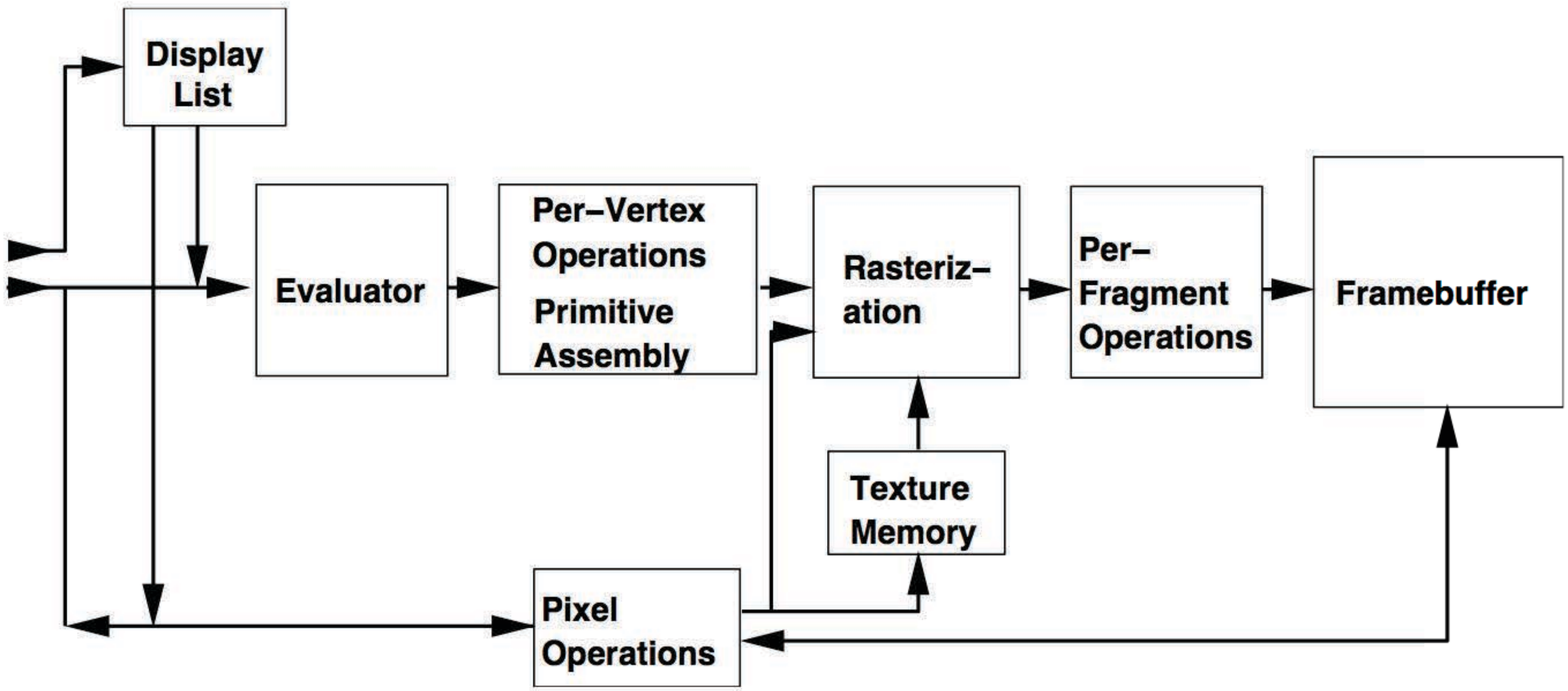
- Download and explore the sample renderer.

1. FORWARD RENDERING

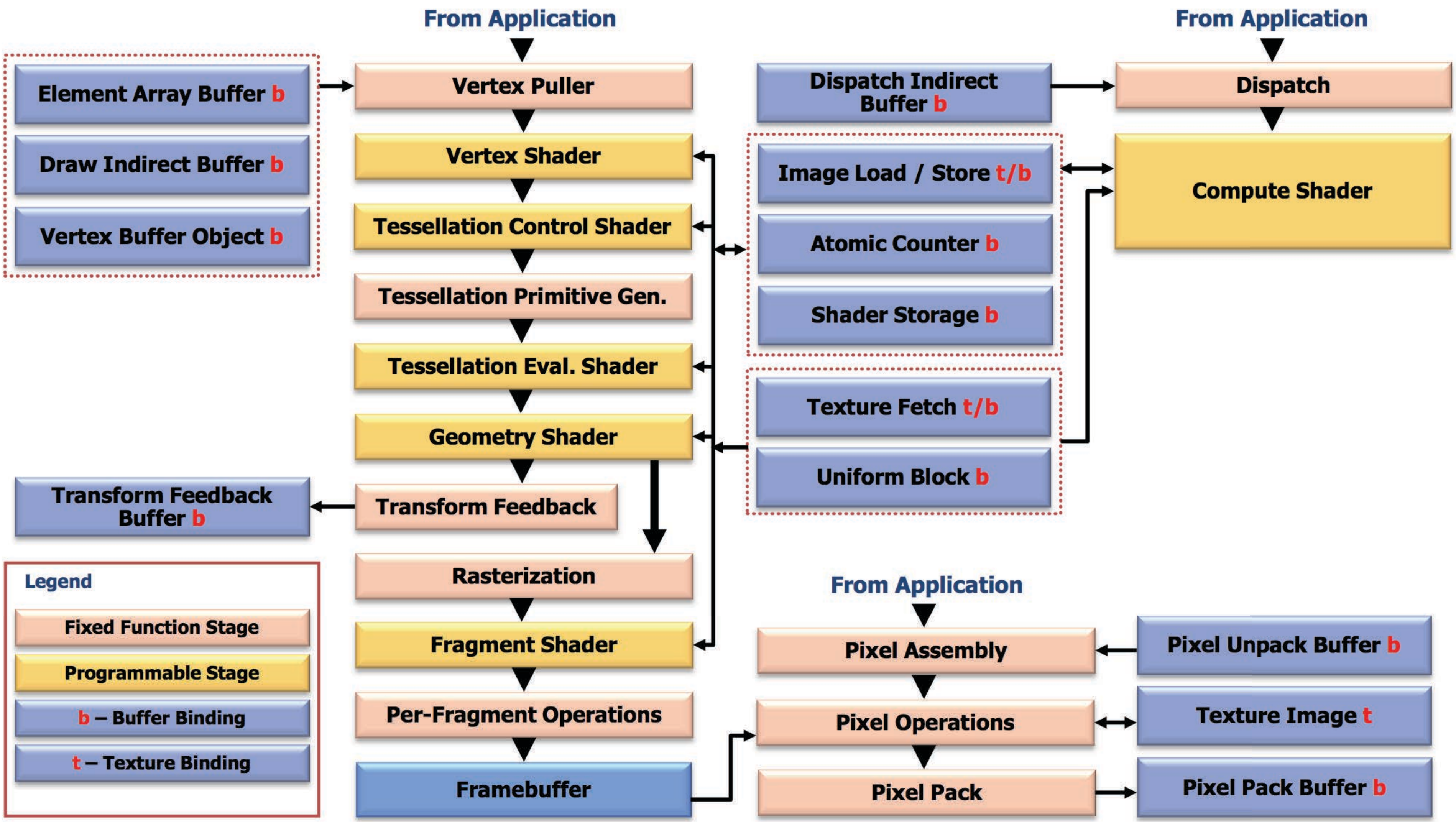
FORWARD RENDERING GENERAL APPROACH

- The "traditional" approach since early OpenGL.

- Geometric objects are sent as primitives to the GL.
- Their vertices are transformed and processed by the vertex shader.
- The primitives are rasterized.
- The fragments are processed by the fragment shader (final color, depth test, masking, blending).
- Shading can happen either in vertex or fragment shader or both.
- Final fragments are written (or not written) to the framebuffer.



Source: opengl.org – OpenGL 1.1 specification



Source: opengl.org – OpenGL 4.4 specification, p32

FORWARD RENDERING OBSERVATIONS & LIMITATIONS

- Classic, widely established and straightforward approach.
- Supported by all graphics hardware.
- *Lighting cost*
  - Every object in the scene is shaded (remember that depth test happens after fragment shading).
  - Complexity is $o(n_{Geometries\_pixels} * n_{Lights})$.
  - Some workarounds do exist (e.g. early depth test).
- Local lighting only, no support for *global illumination* (GI).
- Shader complexity increases with number of geometry types, material types, and light types.

2. DEFERRED RENDERING

DEFERRED RENDERING
GENERAL IDEA

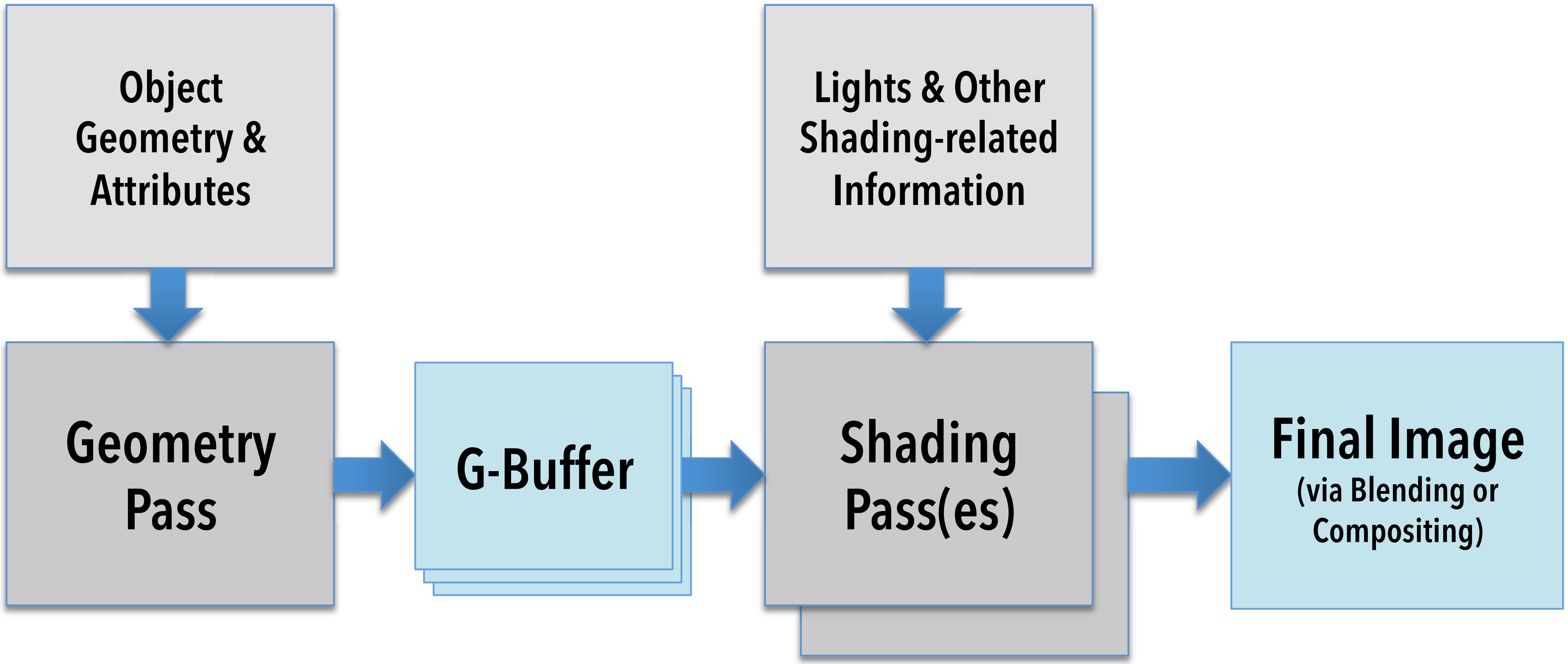
- Multi-pass rendering approaches.
- Emerging methodology, refers to a whole class of approaches, with many different options and possibilities.
- General goal of the approaches is to reduce the amount of fragments shaded (i.e., shade final visible fragments only).

DEFERRED RENDERING ORIGINAL IDEA

- Pass 1: Write geometry plus attributes into buffers (often called *G-Buffers*).
  - Typical attributes: Diffuse color, position, normal, texture coordinates.
- Pass 2 - *n*: Operate on rendered frame buffers (i.e. in screen space). Use attributes to calculate final pixel color.

M Deering, S Winner, B Schediwy, C Duffy, N Hunt (1988). "The triangle processor and normal vector shader: a VLSI system for high performance graphics". ACM SIGGRAPH Computer Graphics 22 (4): 21–30.

T Saito, T Takahashi (1990). "Comprehensible rendering of 3-D shapes". ACM SIGGRAPH Computer Graphics 24 (4): 197–206.

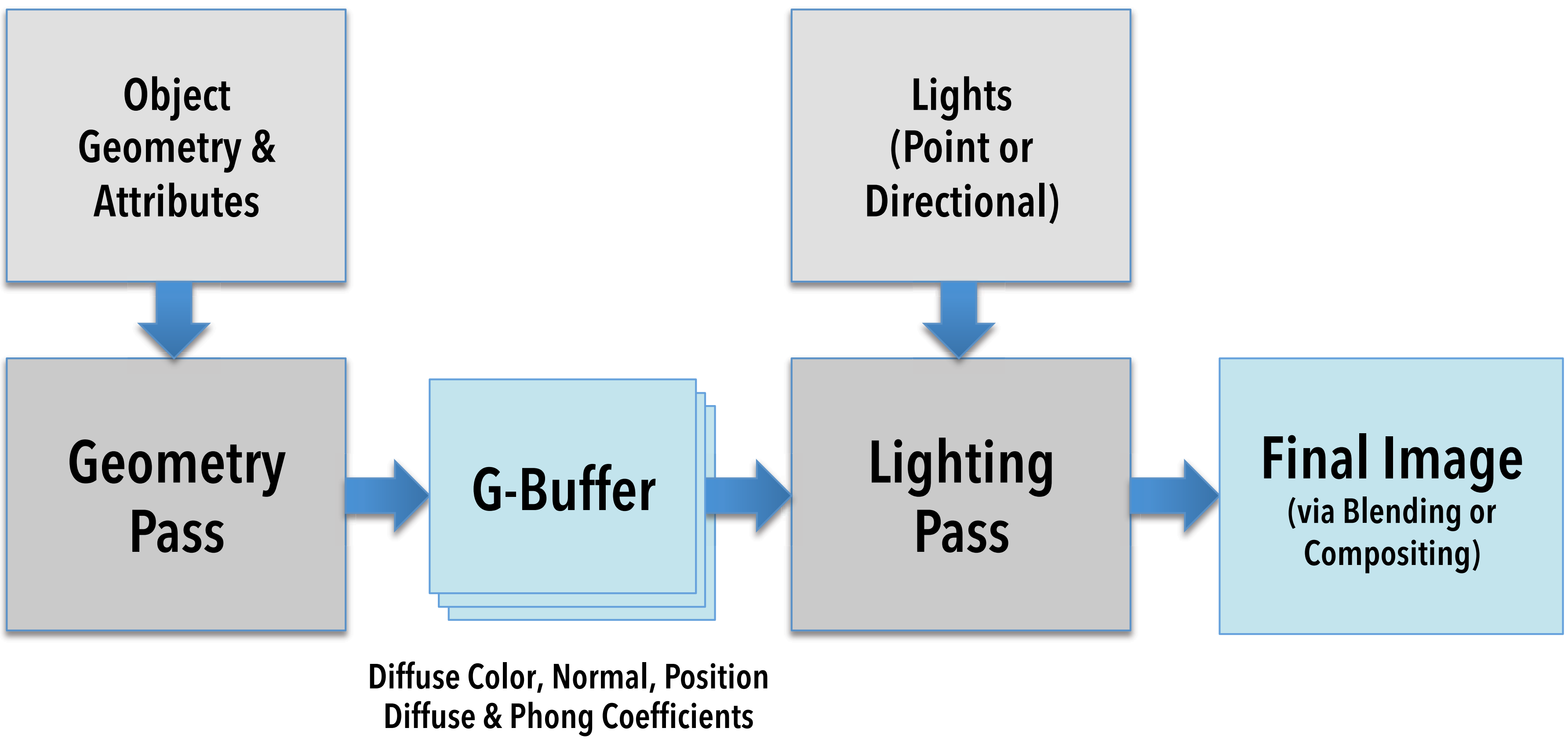

Object
Geometry &
Attributes
Lights & Other
Shading-related
Information
Geometry
Pass
G-Buffer
Shading
Pass(es)
Final Image
(via Blending or
Compositing)

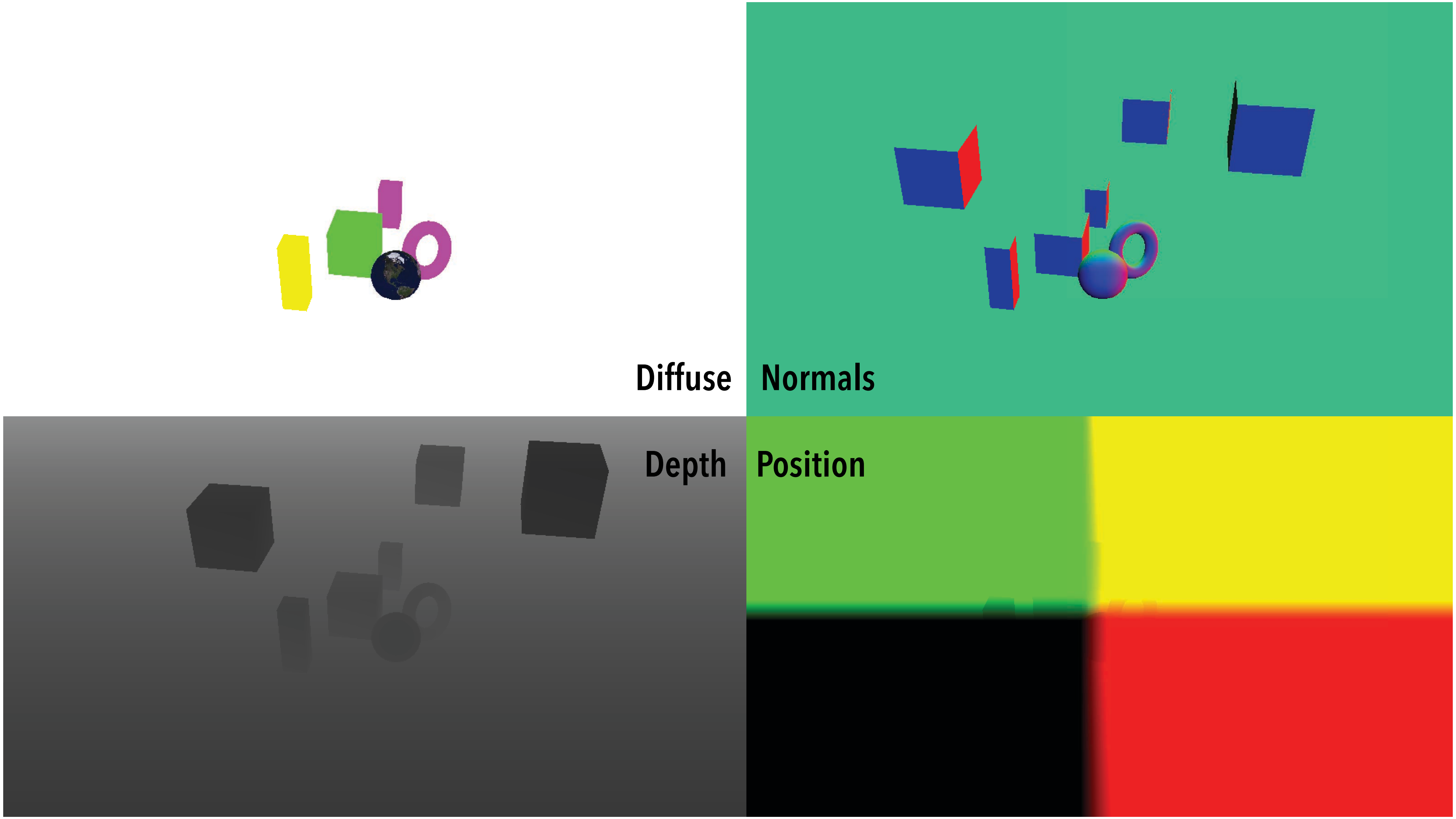
3A. EXAMPLE USES OF DR
DEFERRED SHADING

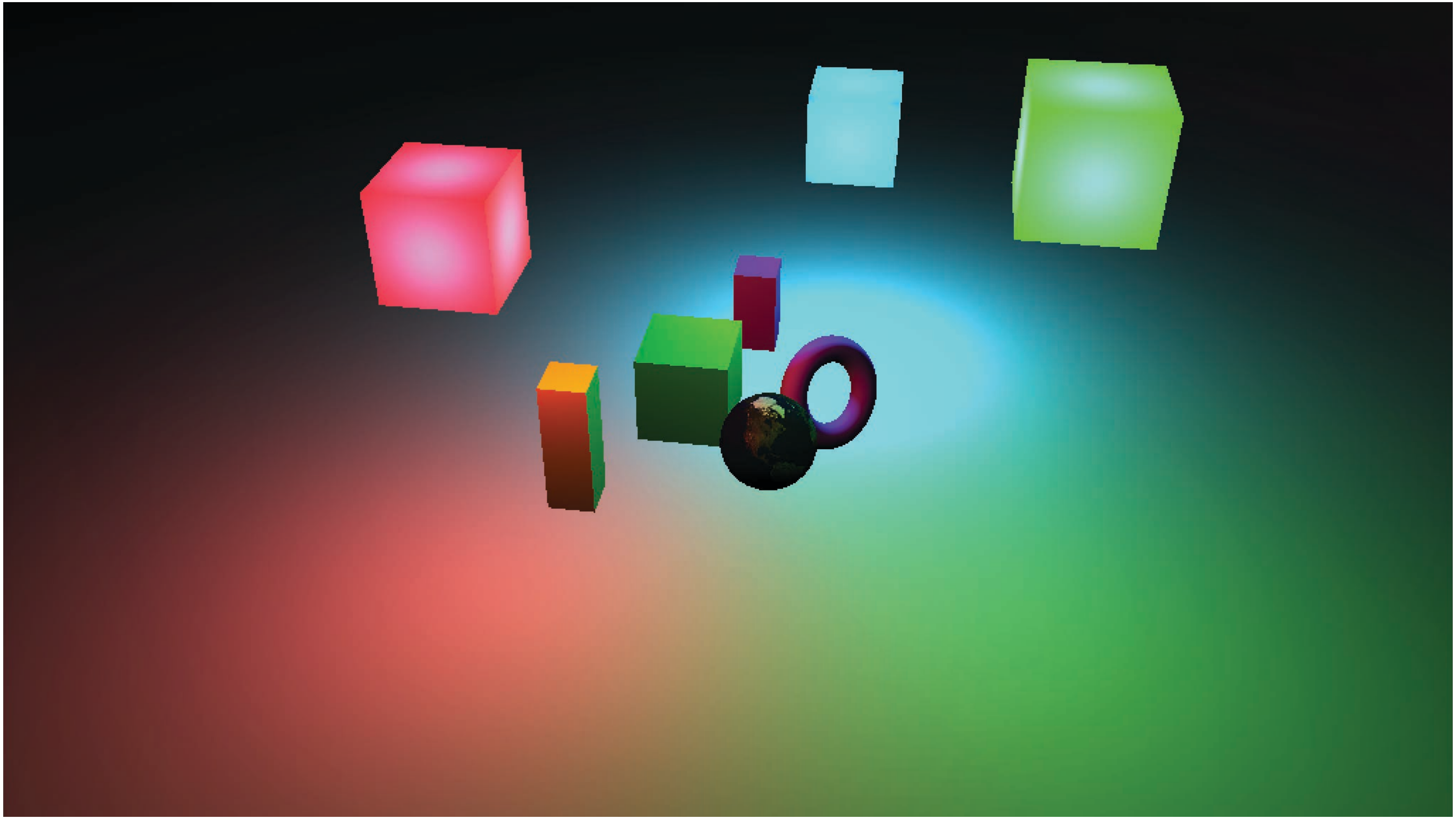
DEFERRED SHADING GENERAL IDEA

- Similar to previously shown diagram.
- Geometry pass stores information required for lighting in each fragment: Diffuse color, position, normals.
- Shading pass then shades each framebuffer pixel for each light according to chosen shading equation.

$$L(\mathbf{v},\mathbf{n}) = \sum_{k=1}^{n} c_{diff} \otimes f_{diff}(B_{L_k}, \mathbf{l}_k, \mathbf{v}, \mathbf{n}) + c_{spec} \otimes f_{spec}(B_{L_k}, \mathbf{l}_k, \mathbf{v}, \mathbf{n})$$


Object
Geometry &
Attributes
Lights
(Point or
Directional)
Geometry
Pass
G-Buffer
Lighting
Pass
Final Image
(via Blending or
Compositing)
Diffuse Color, Normal, Position
Diffuse & Phong Coefficients


Diffuse
Normals
Depth
Position

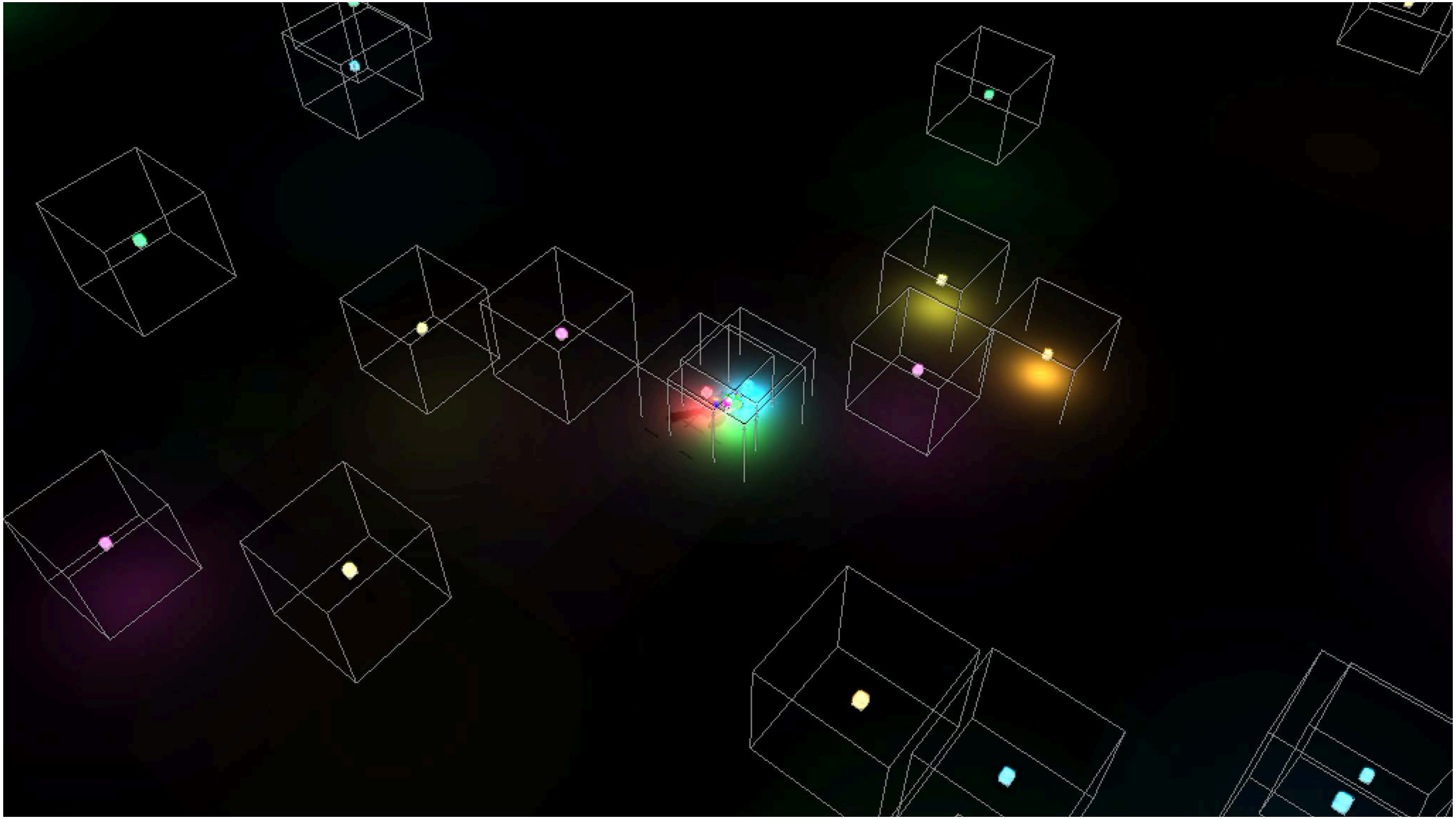


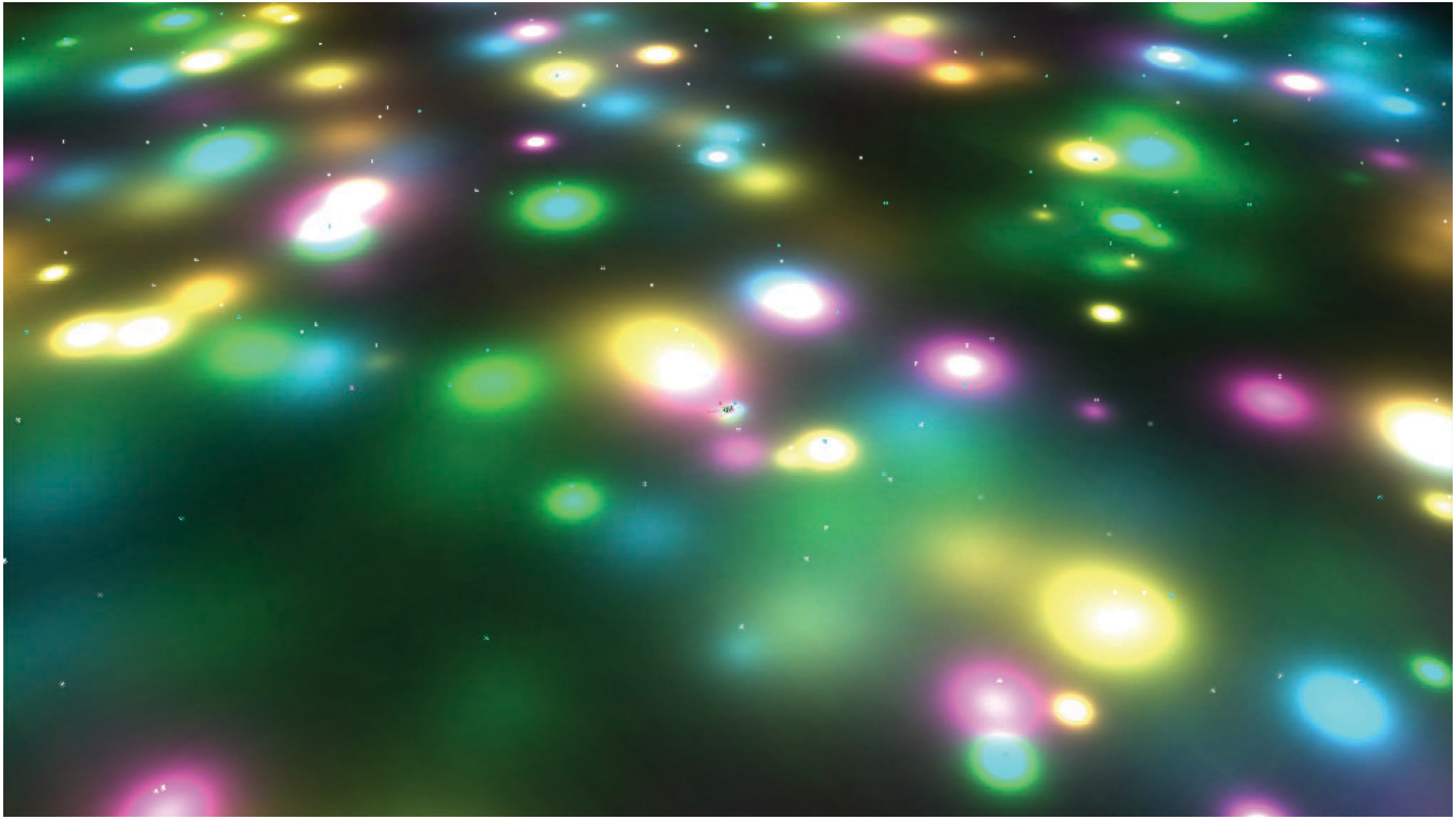
DEFERRED SHADING GENERAL IDEA

- Similar to previously shown diagram
- Geometry pass stores information required for lighting in each fragment: Diffuse color, position, normals.
- Shading pass then shades each framebuffer pixel for each light according to chosen shading equation.
- Overall complexity $o(n_{Framebuffer\_pixels} * n_{Lights})$

DEFERRED SHADING
LIGHT VOLUMES

- Most lights do not influence every pixel, e.g. when distant / small.
- Thus, for each light we can only draw the area of influence.
  - For point lights, use a sphere or a cube.
  - For directional lights, use a cone or a pyramid.
- Size of objects is determined by a light's attenuation factors.





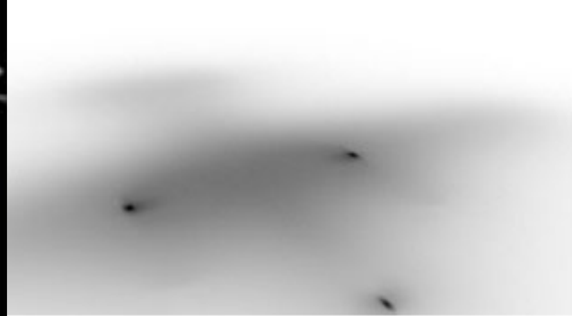
3B. EXAMPLE USES OF DR AMBIENT OCCLUSION

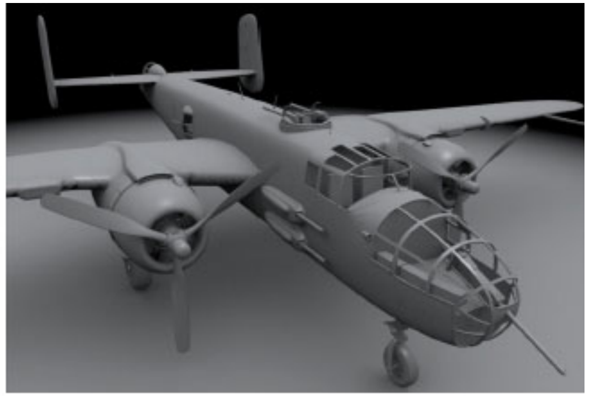
AMBIENT OCCLUSION INTRODUCTION

- Ambient occlusion is a technique that approximates *global illumination (GI)*, and in particular deals with *ambient environment lighting*.
- In simple terms, the techniques takes into account that some areas of an object receive less light from the surrounding environment than others.
- Such areas need to be attenuated.
- First implemented by Hayden Landis et al. at ILM around 2002 for RenderMan, i.e. non-real-time.

Ambient Occlusion

AO Contact Shadow





[*http://renderman.pixar.com/view/production-ready-global-illumination*](http://renderman.pixar.com/view/production-ready-global-illumination)

AMBIENT OCCLUSION EARLIER IMPLEMENTATIONS

- Generate *ambient occlusion map* for the model.
- Render the map together with environment map.
- Generation can be hardware accelerated, e.g., using shadow mapping with a large number of lights.

```
For each triangle {
  Compute center of triangle
  Generate set of rays over the hemisphere there
  Vector avgUnoccluded = Vector(0, 0, 0);
  int numUnoccluded = 0;
  For each ray {
    If (ray doesn't intersect anything) {
      avgUnoccluded += ray.direction;
      ++numUnoccluded;
    }
  }
  avgUnoccluded = normalize(avgUnoccluded);
  accessibility = numUnoccluded / numRays;
}
```

From NVidia GPUGems, Chapter 17
http://http.developer.nvidia.com/GPUGems/gpugems_ch17.html

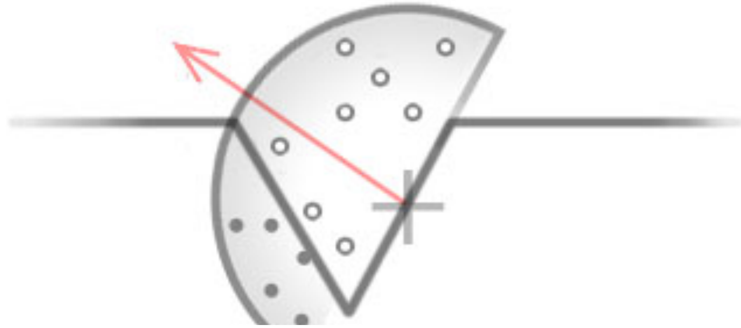
AMBIENT OCCLUSION APPROXIMATION USING DR

How can AO be achieved using deferred rendering?

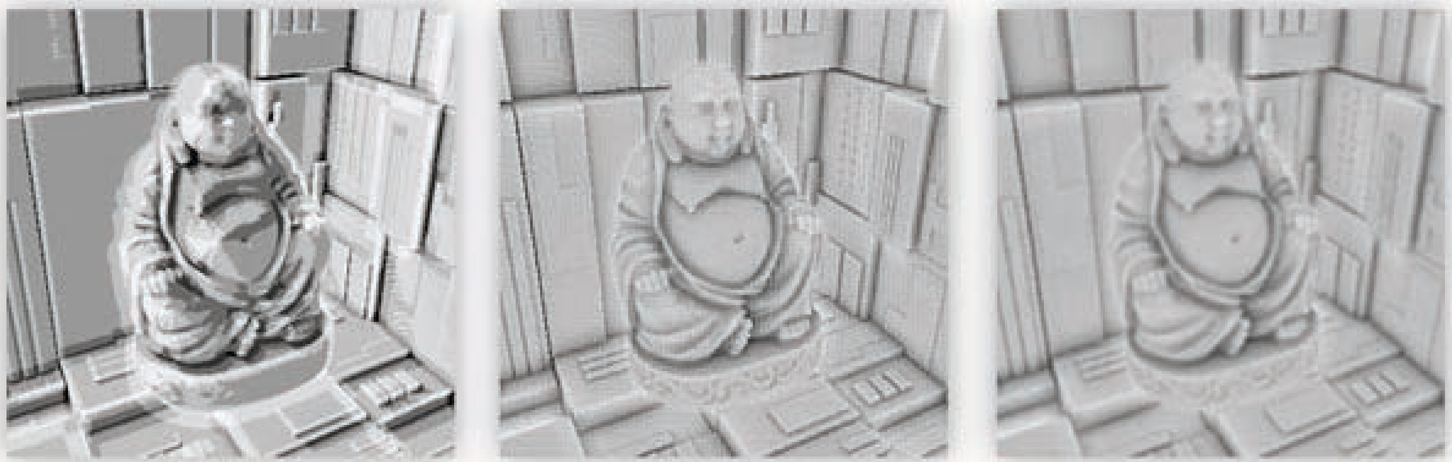
SSAO: SCREEN SPACE AMBIENT OCCLUSION

- Approximates AO for real-time applications using a deferred fragment shader.
- Originally developed at Crytec in 2007 for the game Crysis, and then extended / modified by others.
- Instead of casting rays to obtain occlusion information, the SSAO approach samples the depth buffer.

SSAO



- Most of today’s implementations use a depth and a normal map, with random samples on a hemisphere.
- Each sample is then tested whether it occludes the current pixel or not (depending on depth difference).
- The number of samples needs to be reduced to a minimum (typically 10 - 16) to achieve acceptable performance.
- If for every pixel the same samples are used, “banding” results. Therefore the sample locations are randomly rotated for every pixel.
- The random rotation results in noise, which is then removed through blur.



low sample 'banding' random rotation = noise + blur = acceptable

From http://john-chapman-graphics.blogspot.com/2013/01/ssao-tutorial.html

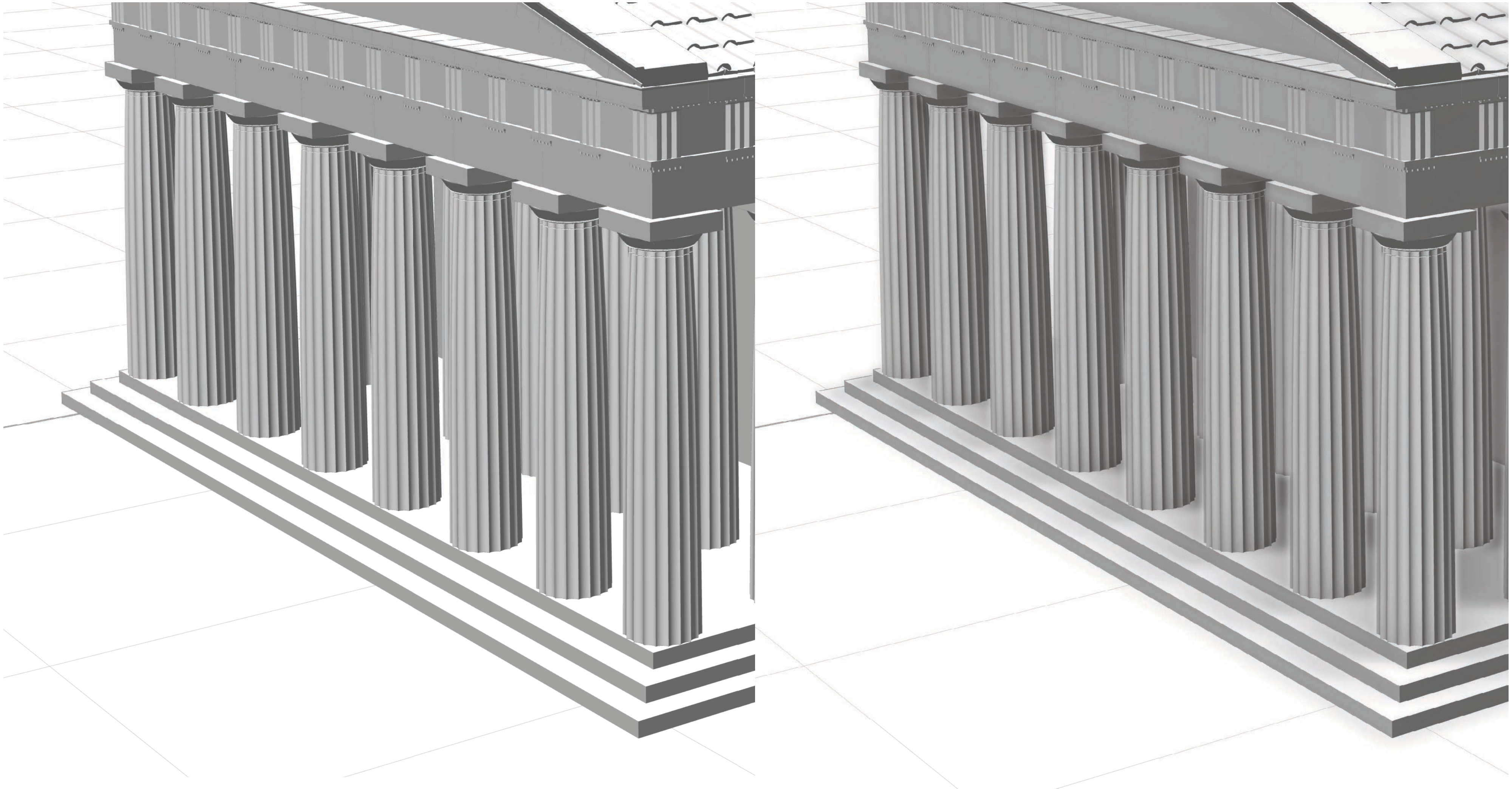
SSAO

Advantages

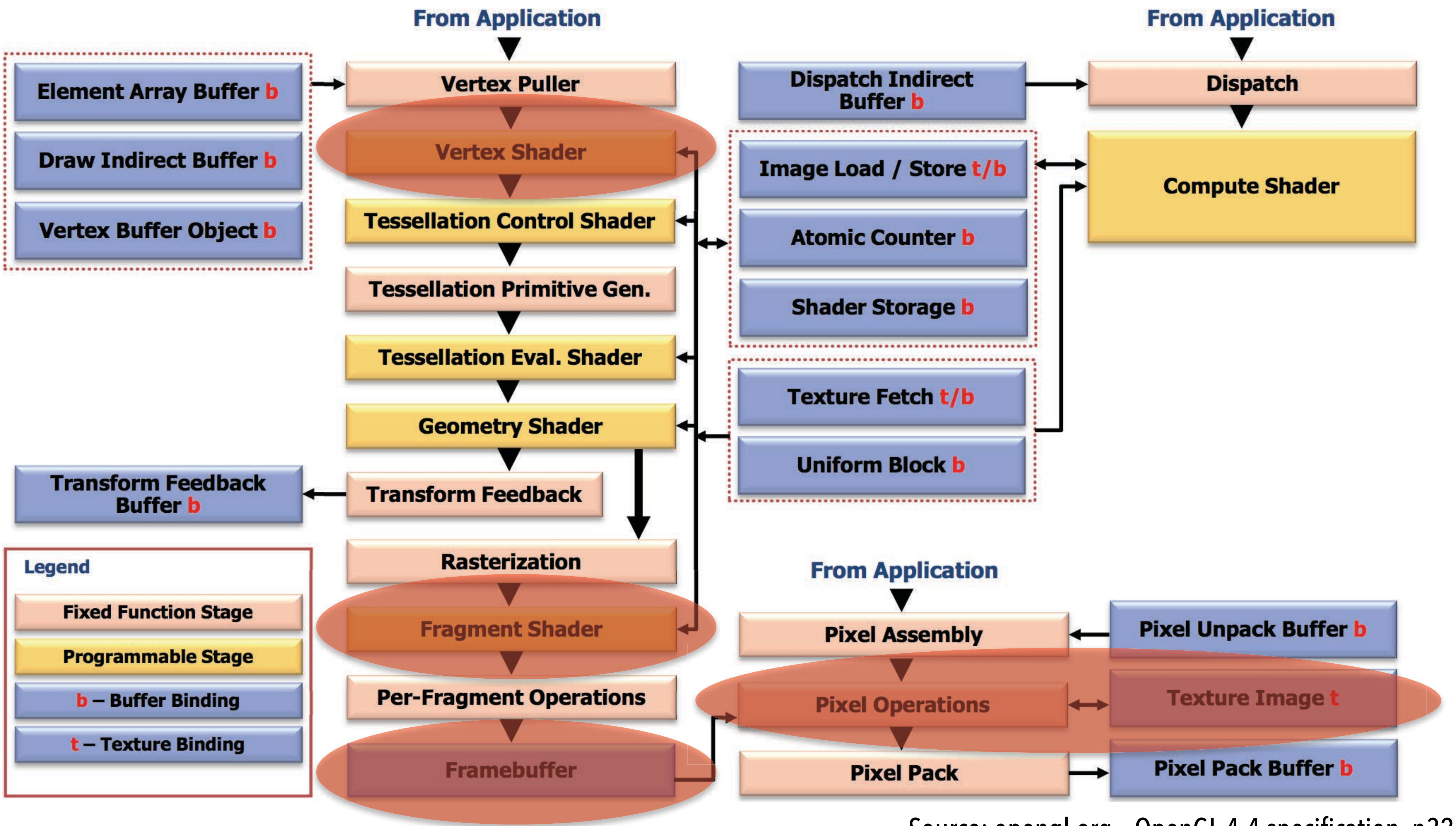
- Independent of scene complexity & dynamics.
- Fully in hardware.
- Well suited for deferred renderers (as normal map is typically available).

Disadvantages

- Noise removal requires extra blur stage.
- Limited “range” of sample sphere makes approach relatively local and view dependent.



4. OPENGL MECHANISMS



Source: opengl.org – OpenGL 4.4 specification, p32

OPENGL MECHANISMS OVERVIEW

OpenGL mechanisms that are used for realizing DR:

1. Programmable shading

   Vertex shading, fragment shading
2. Framebuffer objects (FBOs)
3. Multiple render targets (MRTs)

OPENGL MECHANISMS
FRAMEBUFFER OBJECTS (FBOS)

- FBOs encapsulate a framebuffer that can be used for off-screen rendering.
- Each FBO has a given dimension, and a number of *attachments* (*n* ‘color’ buffers, depth buffer, and stencil buffer).
- Attached buffers are either *textures* or *renderbuffers*.
- FBOs can be enabled for writing and reading.

OPENGL MECHANISMS
FBO RELEVANT API

- `glGenFramebuffer`, `glDeleteFramebuffers`
- `glBindFramebuffer`
  Bind for reading or writing
- `glClearBuffer`
- `glFramebufferTexture2D`
  Attach texture to FBO
- `glCheckFramebufferStatus`
  Important: Check if FBO is correctly set up

OPENGL MECHANISMS
MULTIPLE RENDER TARGETS (MRT)

- Since we want to write multiple object attributes to the FBO's attachments, the corresponding outputs need to be declared in the shader:

```
out vec4 color;
out vec4 normal;
```

- Use `glDrawBuffers` to enable writing to selected FBO attachments.
- Note: Make sure these specifications match the FBO structure.

5. A SIMPLE DEFERRED RENDERER IN C++/CINDER

CINDER DEFERRED RENDERER OVERVIEW

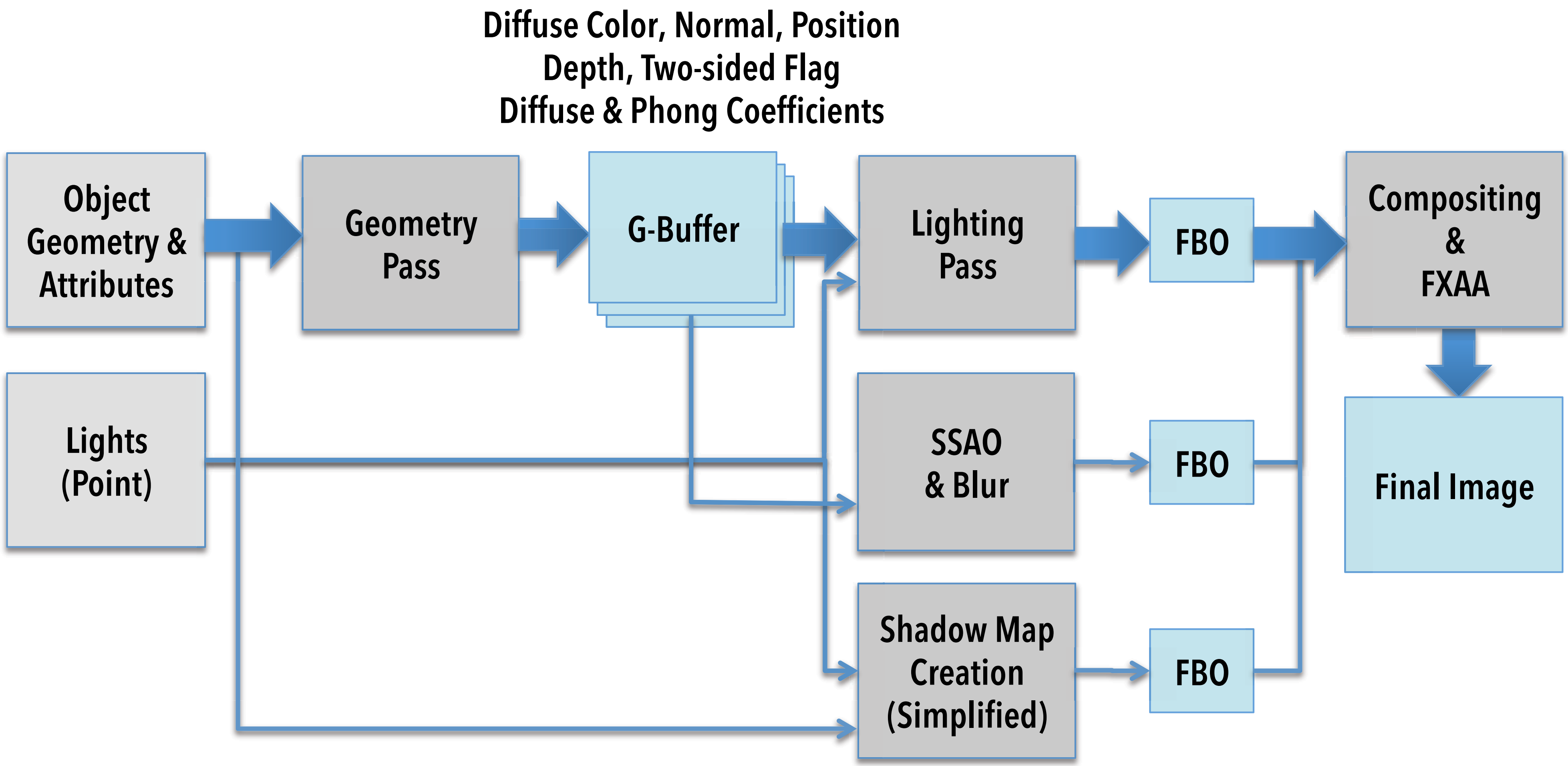
- A small renderer exemplifying some of the presented techniques.
- Written in C++ using the Cinder framework.
- Adapted from original code by Anthony Scavarelli and others, with several fixes and optimizations.

- http://libcinder.org
- https://github.com/arisona/cinder_deferred_renderer

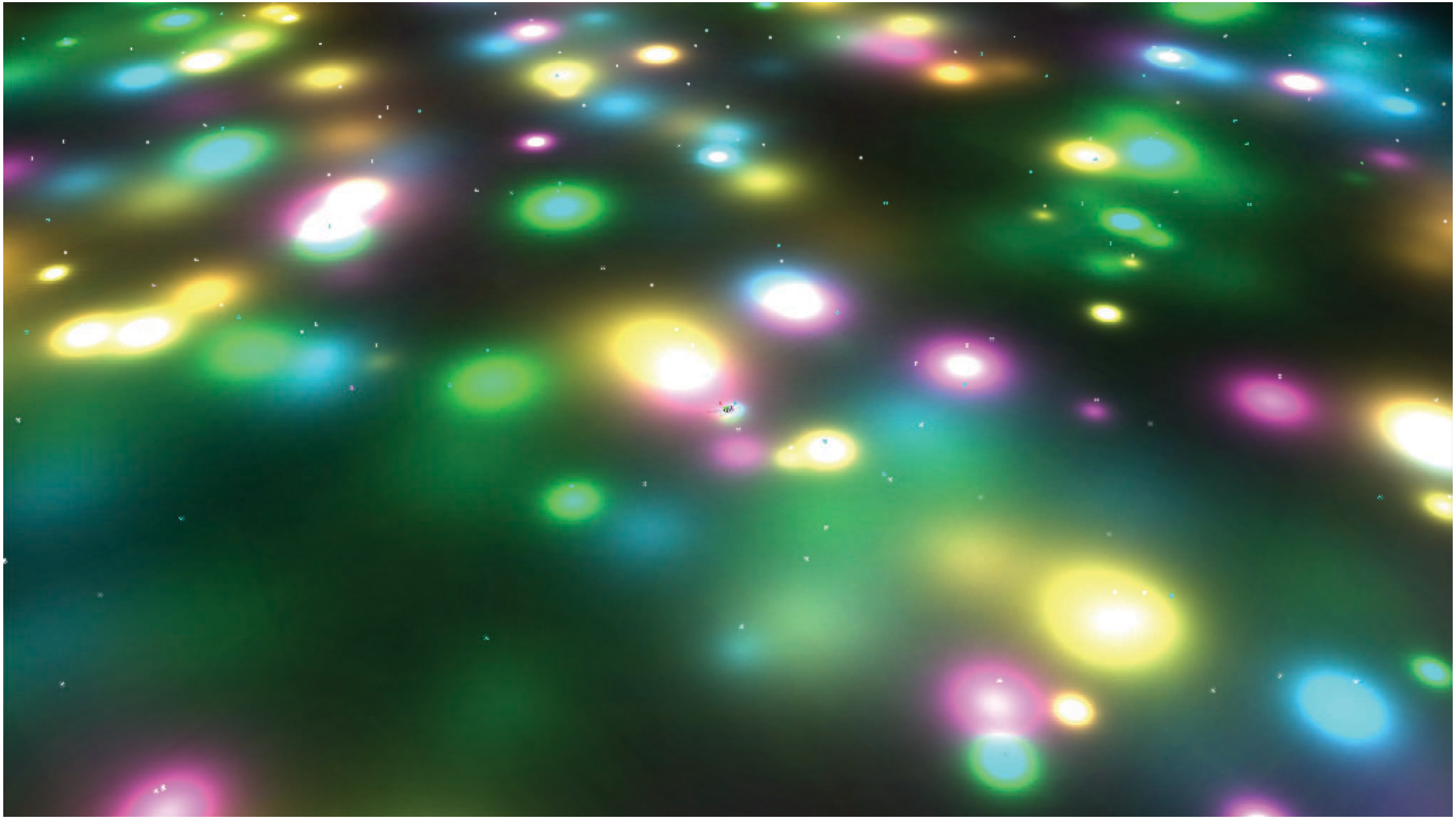
- Note 1: Based on OpenGL 2.0 + Extensions ☹
- Note 2: Updated to use C++11 ☺

CINDER DEFERRED RENDERER FEATURES

- Deferred shading of a large number of point lights (1000+).
- SSAO
  http://www.gamerendering.com/2009/01/14/ssao/
- Shadow support, but not for 1000 lights… (more on this during the next lecture).
- FXAA (screen space software approximation to antialiasing).
  http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf



Diffuse Color, Normal, Position
Depth, Two-sided Flag
Diffuse & Phong Coefficients
Object Geometry & Attributes
Geometry Pass
G-Buffer
Lighting Pass
FBO
Compositing & FXAA
Lights (Point)
SSAO & Blur
FBO
Final Image
Shadow Map Creation (Simplified)
FBO



6. WRAP UP & DISCUSSION

WRAP UP

- DR can be used to build a multi-pass renderer that includes lighting, GI and effect compositing.
- Allows for large amount of (dynamic) lights. However, minimization of light volumes is essential.
- Many game engines include deferred renderers (e.g. Unity Pro, Torque).
- In particular, SSAO has become a standard technique for ambient lighting.

WRAP UP
DR LIMITATIONS

- Requires reasonably modern, programmable graphics hardware (not generally a problem today).
- Memory usage.
- Memory bandwidth.
- No support for transparency, need to use forward rendering for semi-transparent objects.
- If above limitations become a factor, forward rendering can still be the better choice.

UNITY RENDERING PATHS

Rendering Paths Comparison

	Deferred Lighting	Forward Rendering	Vertex Lit
Features			
Per-pixel lighting (normal maps, light cookies)	Yes	Yes	-
Realtime shadows	Yes	1 Directional Light	-
Dual Lightmaps	Yes	-	-
Depth&Normals Buffers	Yes	Additional render passes	-
Soft Particles	Yes	-	-
Semitransparent objects	-	Yes	Yes
Anti-Aliasing	-	Yes	Yes
Light Culling Masks	Limited	Yes	Yes
Lighting Fidelity	All per-pixel	Some per-pixel	All per-vertex
Performance			
Cost of a per-pixel Light	Number of pixels it illuminates	Number of pixels * Number of objects it illuminates	-
Platform Support			
PC (Windows/Mac)	Shader Model 3.0+	Shader Model 2.0+	Anything
Mobile (iOS/Android)	OpenGL ES 2.0	OpenGL ES 2.0	OpenGL ES 2.0 & 1.1
Consoles	360, PS3	360, PS3	-

Page last updated: 2013-08-27

Source: Unity Technologies

7. QUESTIONS & LINKS

LINKS: DEFERRED SHADING

- http://en.wikipedia.org/wiki/Deferred_shading
- http://ogldev.atspace.co.uk/www/tutorial35/tutorial35.html

  OpenGL deferred shading tutorial (tutorials 35 - 37).
- http://developer.amd.com/wordpress/media/2012/10/Deferred%20Shading%20Optimizations.pps

  Excellent discussion on deferred shading strategies and optimizations.
- http://www.realtimerendering.com/blog/deferred-lighting-approaches/

  Some additional considerations regarding deferred lighting.
- http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf

  FXAA method used in sample code. Very useful for screen space operations.

LINKS: AO / SSAO

- http://en.wikipedia.org/wiki/Ambient_occlusion
- http://renderman.pixar.com/view/production-ready-global-illumination

  Good overview into ambient lighting techniques by Hayden Landis.
- http://http.developer.nvidia.com/GPUGems/gpugems_ch17.html

  Overview of AO (not SSAO) implementation of GPUs.
- http://en.wikipedia.org/wiki/Screen_space_ambient_occlusion
- http://john-chapman-graphics.blogspot.com/2013/01/ssao-tutorial.html

  Comprehensive and intuitive SSAO tutorial.