An Overview of Game Engine Architecture

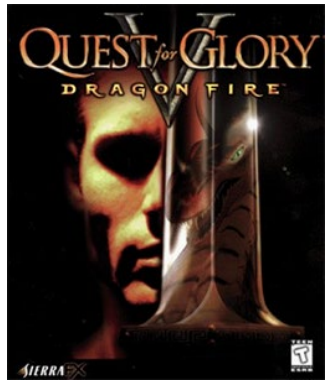
Eric Lengyel, Ph.D.

Virginia Tech

November 5, 2024

Who am I?

- Virginia Tech alumnus, 1991–1996



- Lead programmer for Quest for Glory V at Sierra, 1996–1998
- Worked on OpenGL implementation at Apple, 1999–2000

OpenGL.

- Worked on PlayStation 3 system software at Naughty Dog, 2004–2005
  - Wrote the graphics driver for the PS3

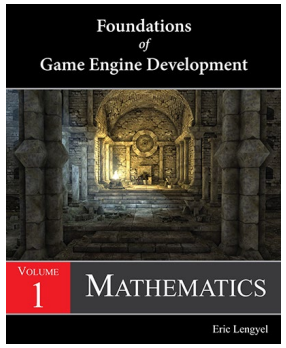
NAUGHTY DOG

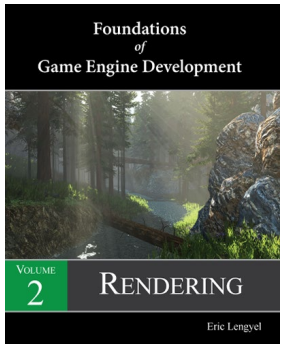
- Running my own business called Terathon Software, 2006–present

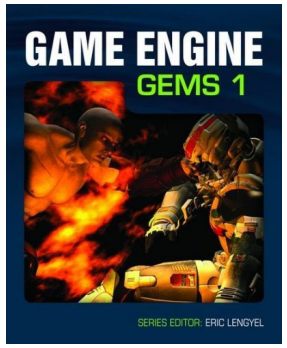
TERATHON SOFTWARE

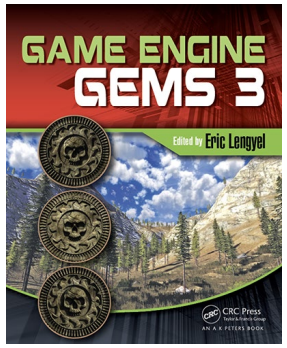
Books / Teaching

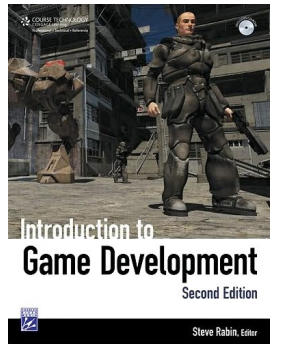
- Written or contributed to many books about game development
- Taught real-time rendering and game engine courses at UCSC

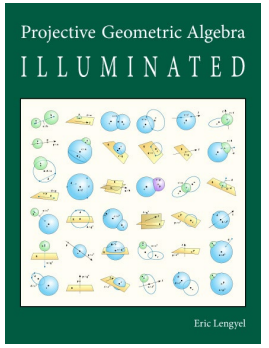


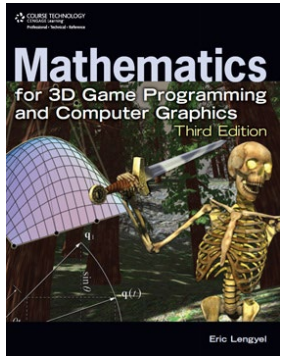


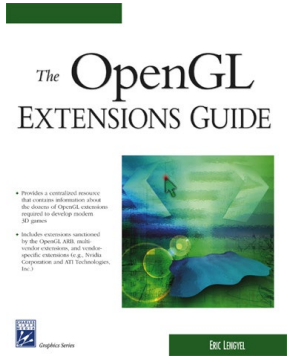


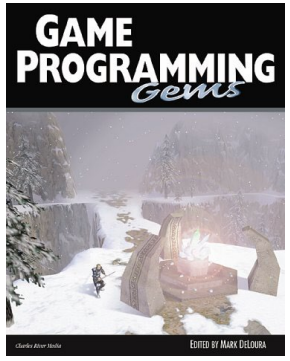


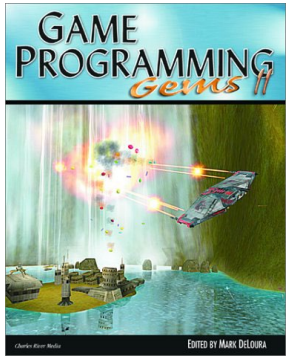


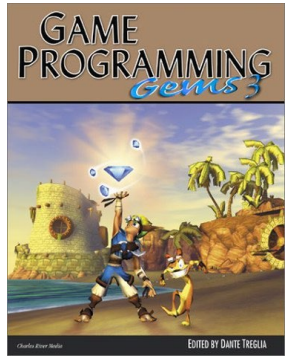


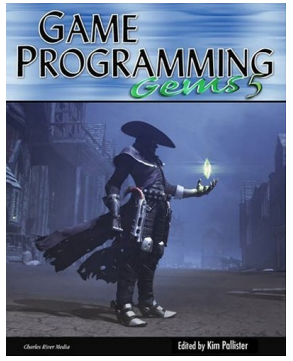






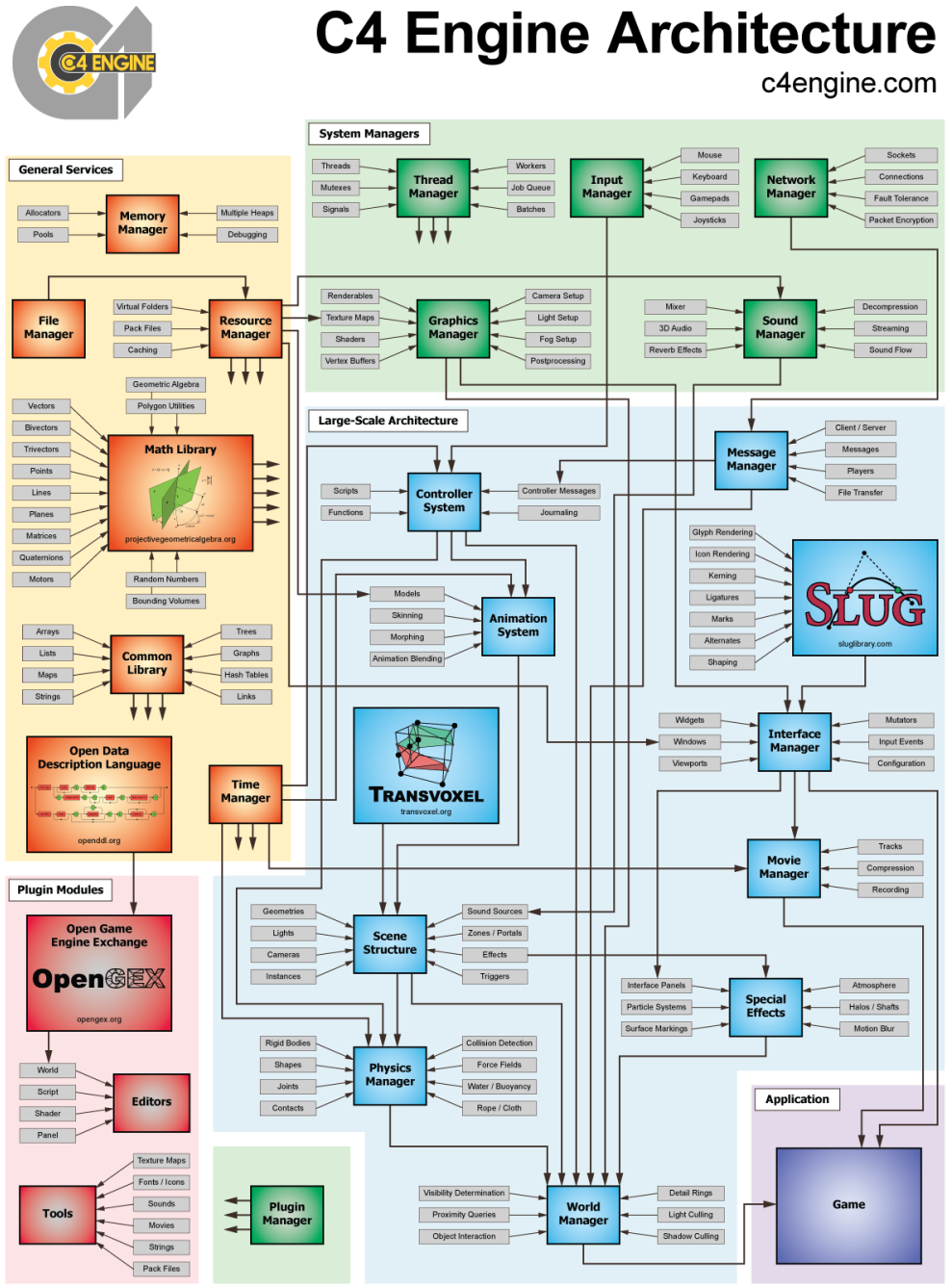






C4 Engine

- Started in 1999
- 619 source files
- 650,000 lines of code (C++)
- Used by several PC and PlayStation games



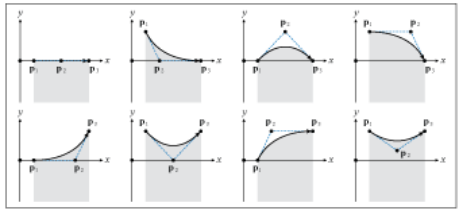
The Slug Algorithm

sluglibrary.com

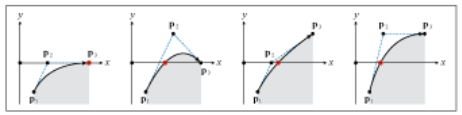
Root Eligibility

Class	$y_3 < 0$	$y_2 < 0$	$y_1 < 0$	Root 2	Root 1
A	0	0	0	0	0
B	0	0	1	1 ●	0
C	0	1	0	1 ●	1 ●
D	0	1	1	1 ●	0
E	1	0	0	0	1 ●
F	1	0	1	1 ●	1 ●
G	1	1	0	0	1 ●
H	1	1	1	0	0
				0x2E	0x74

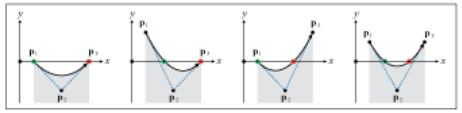
Class A



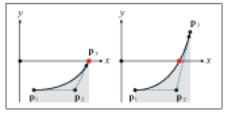
Class B



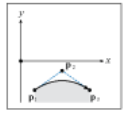
Class C



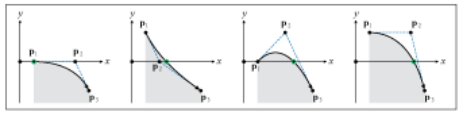
Class D



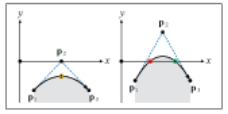
Class H



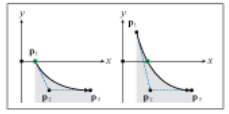
Class E



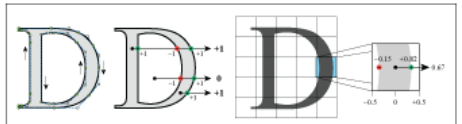
Class F



Class G

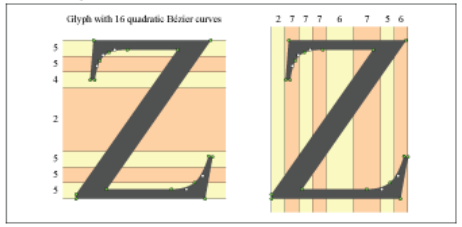


Winding Number

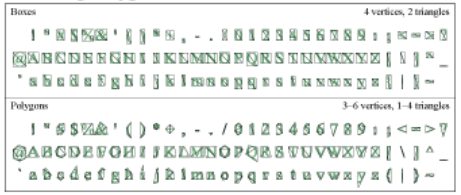


Quadratic Bézier curve	$\mathbf{p}(t) = (1-t)^2\mathbf{p}_1 + 2(1-t)\mathbf{p}_2 + t^2\mathbf{p}_3$	$\mathbf{p}_i = (x_i, y_i)$
Ray intersection equation	$p_y(t) = (y_1 - 2y_2 + y_3)t^2 - 2(y_1 - y_2)t + y_1 = 0$	
Potential solutions	$t_1 = \frac{b - \sqrt{b^2 - ac}}{a}$ $t_2 = \frac{b + \sqrt{b^2 - ac}}{a}$	$\frac{d}{dt}p_y(t_1) \le 0$ $\frac{d}{dt}p_y(t_2) \ge 0$
	$a = y_1 - 2y_2 + y_3$ $b = y_1 - y_2$ $c = y_1$	
Change to winding number for ray in positive x direction	$+\text{sat}(k p_x(t_1) + \frac{1}{2})$ if root 1 eligible $-\text{sat}(k p_x(t_2) + \frac{1}{2})$ if root 2 eligible	k = pixels per em
Change to winding number for ray in negative x direction	$-\text{sat}(\frac{1}{2} - k p_x(t_1))$ if root 1 eligible $+\text{sat}(\frac{1}{2} - k p_x(t_2))$ if root 2 eligible	

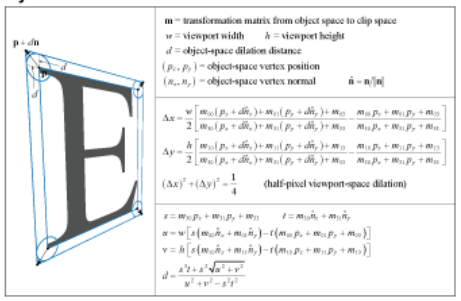
Geometry Bands



Bounding Polygons



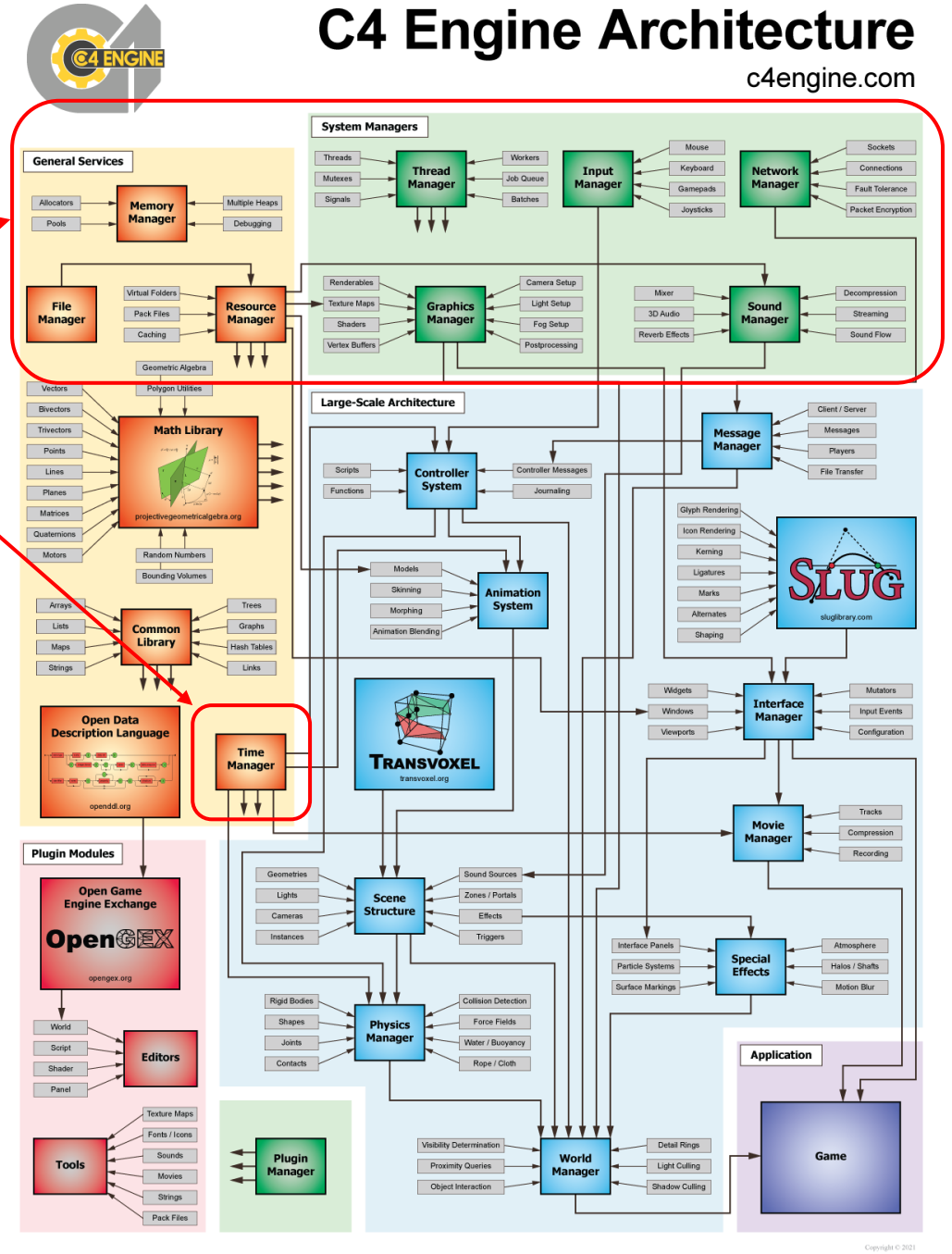
Dynamic Dilation



- Slug is a GPU-centered font and vector graphics rendering library

These interact with operating system

Everything else is platform agnostic



Lower level

Higher level

General Services

- Low-level services that are used throughout the engine
- Memory manager
  - Special allocators optimized for specific uses
  - Debugging capabilities
- File manager
  - Uses native OS calls for file access
- Resource manager
  - Organization into virtual folders / pack files
  - Cache management
- Time manager
  - Uses native OS calls to get precise timestamps

General Services

- Math library
  - Vectors, points, lines, planes, matrices, quaternions, Bézier curves, colors, polygons, boxes, random numbers, ...
- Common utility library
  - Arrays, lists, maps, trees, graphs, hash tables, strings, ...
- https://github.com/EricLengyel/Terathon-Math-Library
- https://github.com/EricLengyel/Terathon-Container-Library

Generic Data Format

- Something for config files, key bindings, import settings, ...
- Could use JSON, XML, etc.

- C4 uses the Open Data Description Language (OpenDLL)
  - https://openddl.org/
  - https://github.com/EricLengyel/OpenDDL
- This arose during development of the Open Game Engine Exchange format (OpenGEX)

System Managers

- Low-level interface with OS for various hardware access
- Thread Manager
  - Handles job queues for multiple CPU cores
- Input Manager
  - Takes care of keyboard, mouse, game controllers, joysticks, steering wheels, or other kinds of USB devices
- Sound Manager
  - Plays sound effects, often with 3D localization
  - Streams / decompresses music
- Network Manager
  - Handles low-level internet connections, often using UDP

Graphics Manager

- Provides engine-level interface to native graphics API
  - Direct3D, Vulkan, OpenGL, console-specific

- Handles drawing, render state, shaders, textures, etc.
- Draws to render targets, handles postprocessing effects
- Contains camera, projection matrix setup
- Contains low-level code for light sources and shadows

Large-Scale Architecture

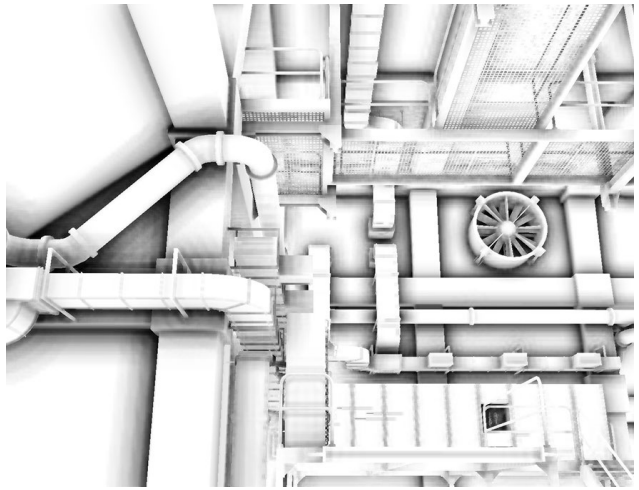
- Controller system
  - In charge of just about anything that moves
- Animation system
  - Handles character animation, blending, morphing, skinning
- Interface manager
  - All things GUI, widgets, windows, buttons, typography, ...
- Physics manager
  - Rigid bodies, joints, collision detection, force fields, buoyancy, ...
- World manager
  - Visibility determination, high-level scene structure

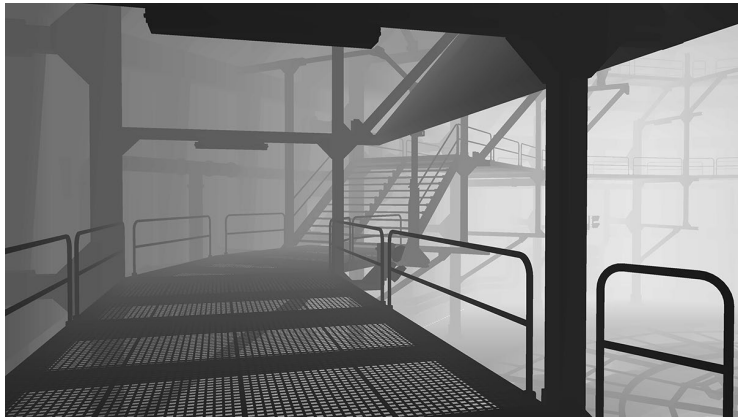
Plugins

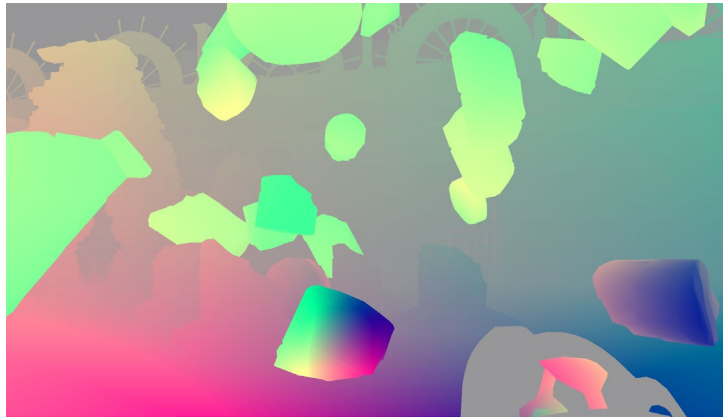
- Tools that help with game development
- World editor
- GUI editor
- Script editor
- Shader editor
- Import code for creating textures, fonts, audio, etc.
- Import code for bringing models into world editor
  - C4 uses Open Game Engine Exchange (OpenGEX) or Collada

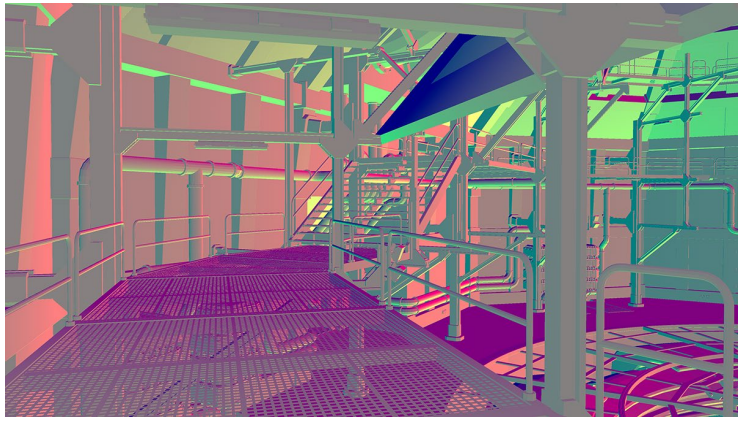
Render Targets

- Linear depth
- Gradient
- Ambient occlusion
- Velocity
- Distortion
- Glow/bloom
- Atmosphere



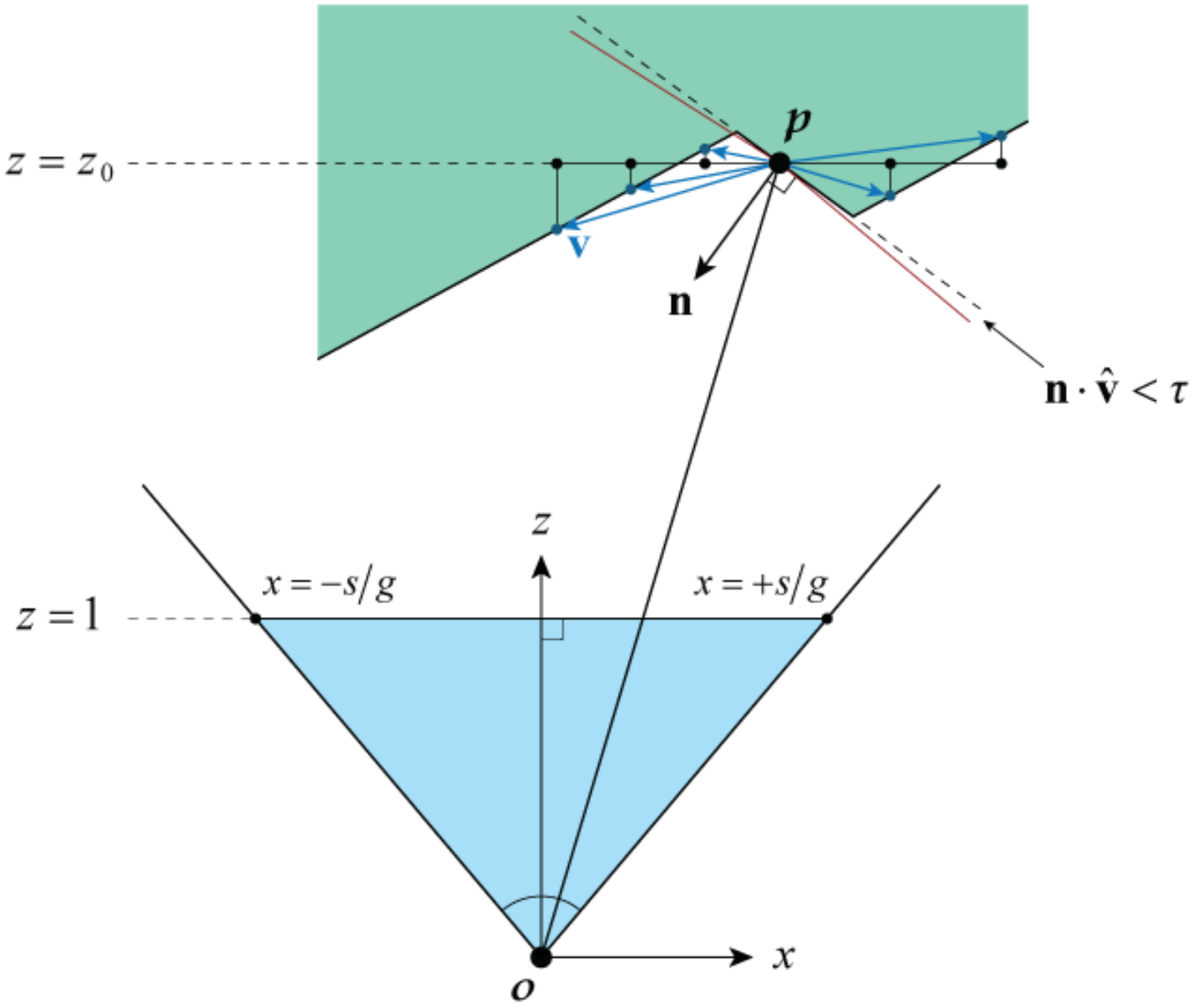




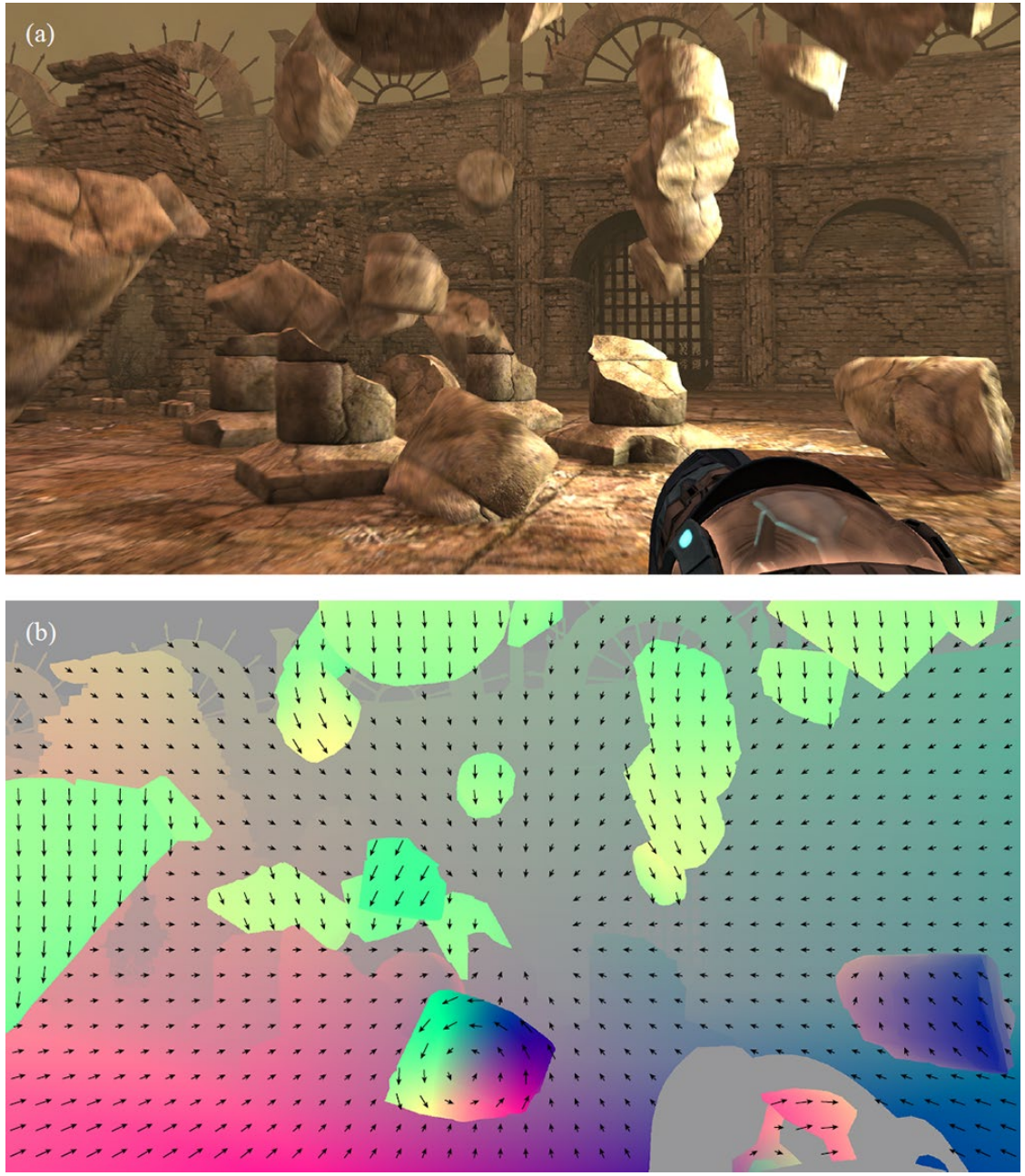


Screen-Space Ambient Occlusion

SSAO

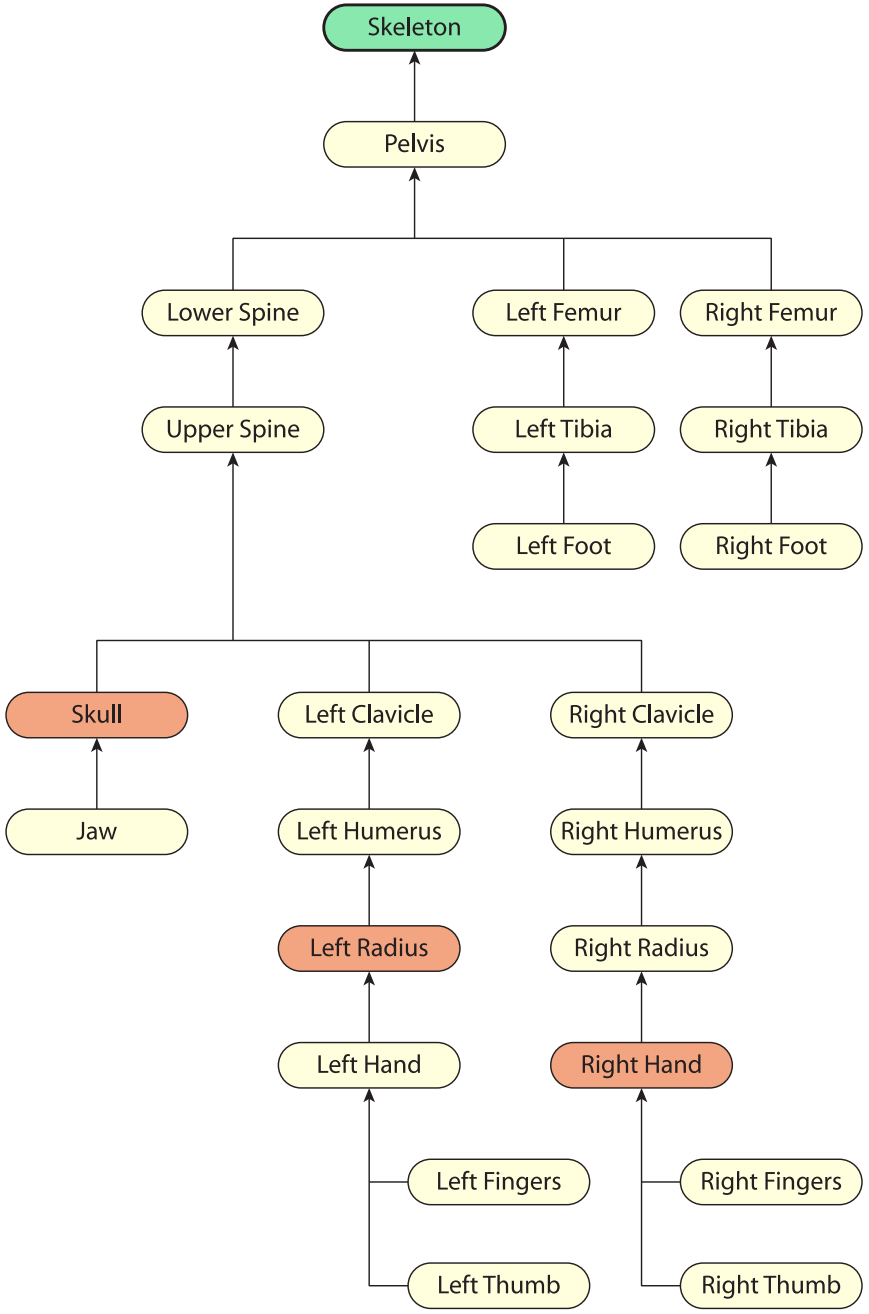

$z = z_0$
p
$\mathbf{v}$
$\mathbf{n}$
$\mathbf{n} \cdot \hat{\mathbf{v}} < \tau$
z
$x = -s/g$
$x = +s/g$
$z = 1$
o
x

Velocity Buffer / Motion Blur



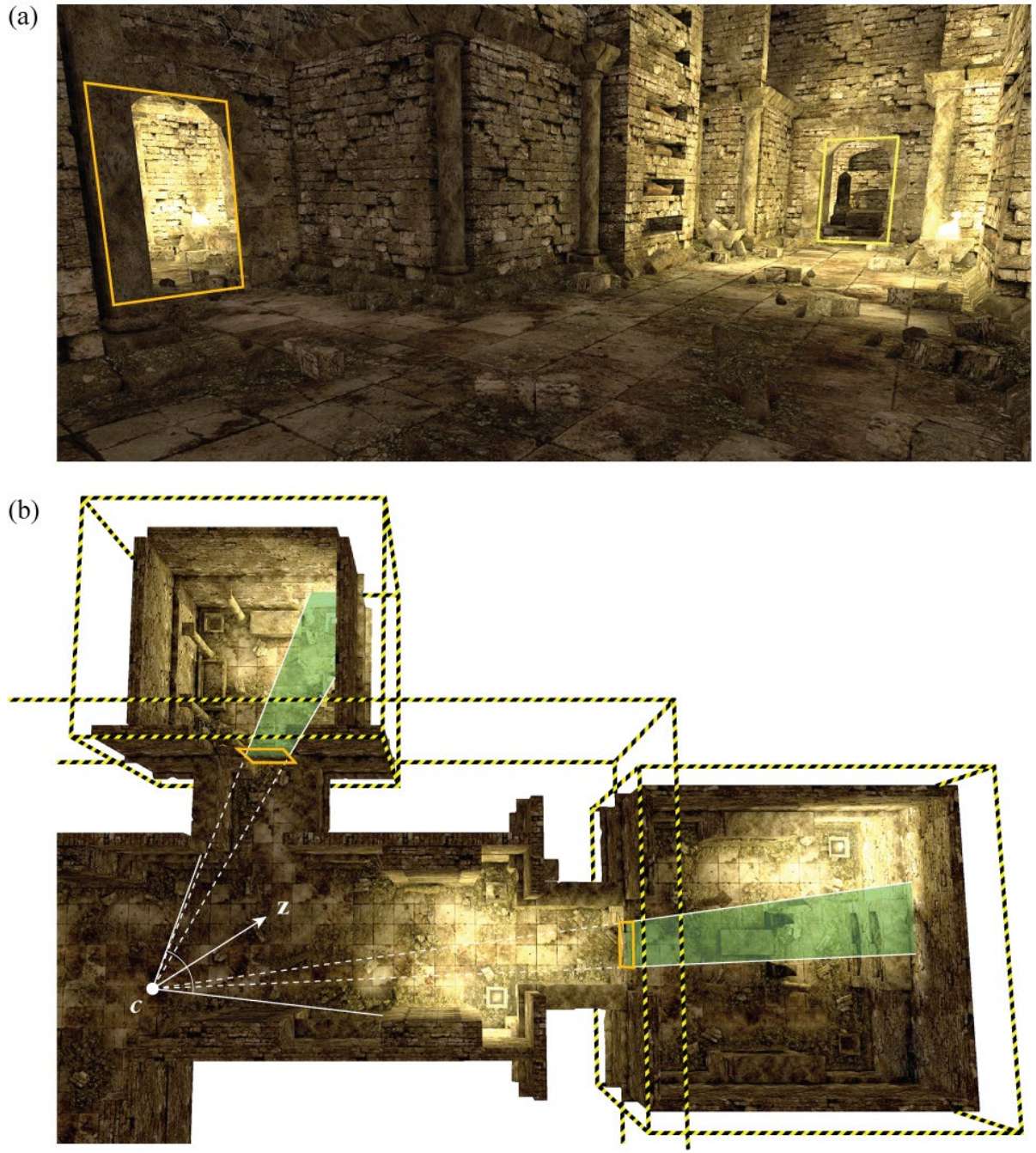
Atmospheric Shadowing

Node Hierarchy



Portal Systems

- World is divided into *zones*
- Zones are connected by *portals*
- A portal is a convex polygon
  - Wound CCW from front side
  - One way window into neighboring zone

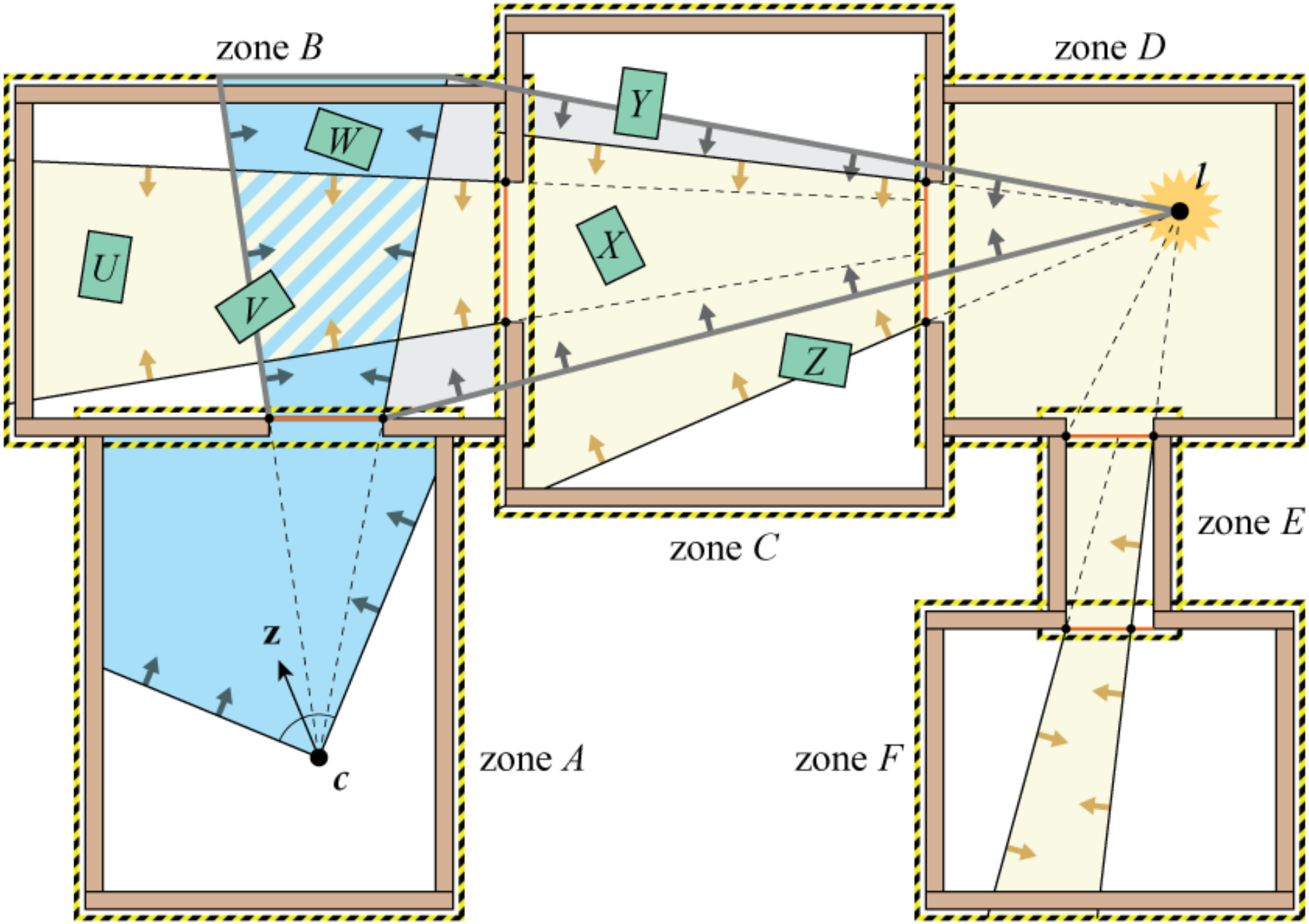

(a)
(b)
z
c

Visibility Regions

- Portals are clipped against planes of view frustum
- Clipped polygon is extruded to create new set of lateral planes
  - Can be capped with near and far planes
- This is a convex region of space called a visibility region

Light Regions

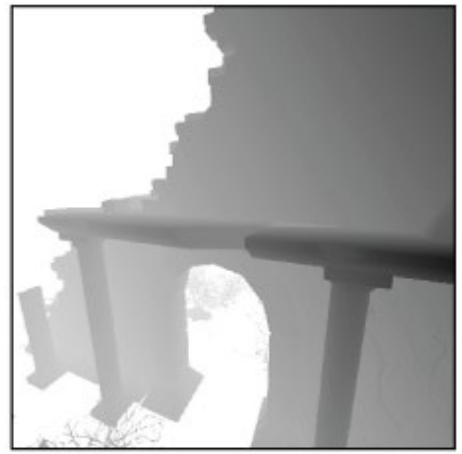
- Portals can also be used to determine where light reaches
- Creates a tree of light regions
- Intersection with visibility regions tells us what objects to light


zone B
zone D
zone C
zone E
zone A
zone F
W
Y
X
U
V
Z
l
z
c

2D Shadow Maps

(a)

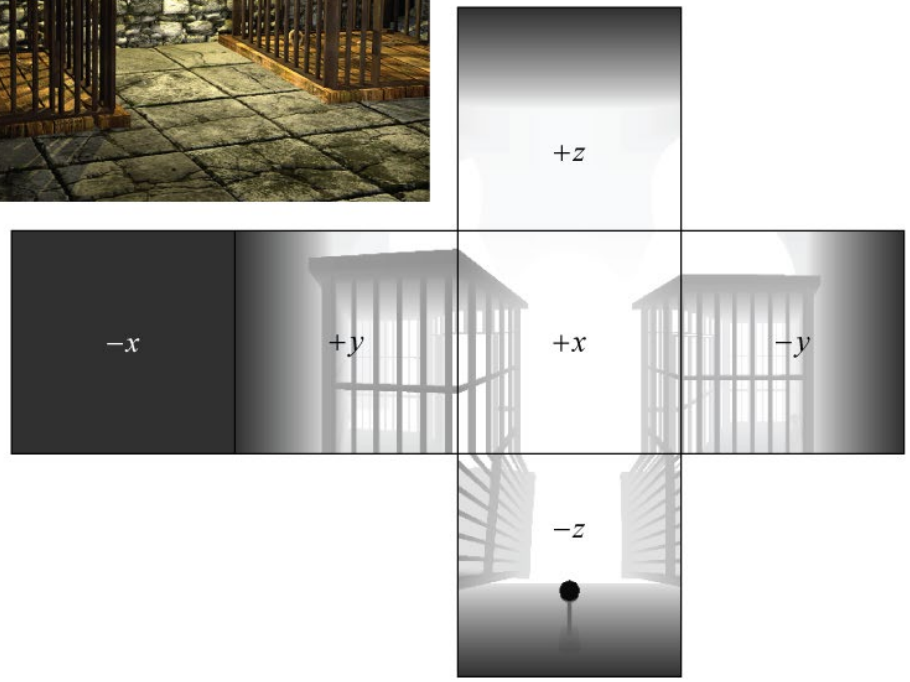
(b)



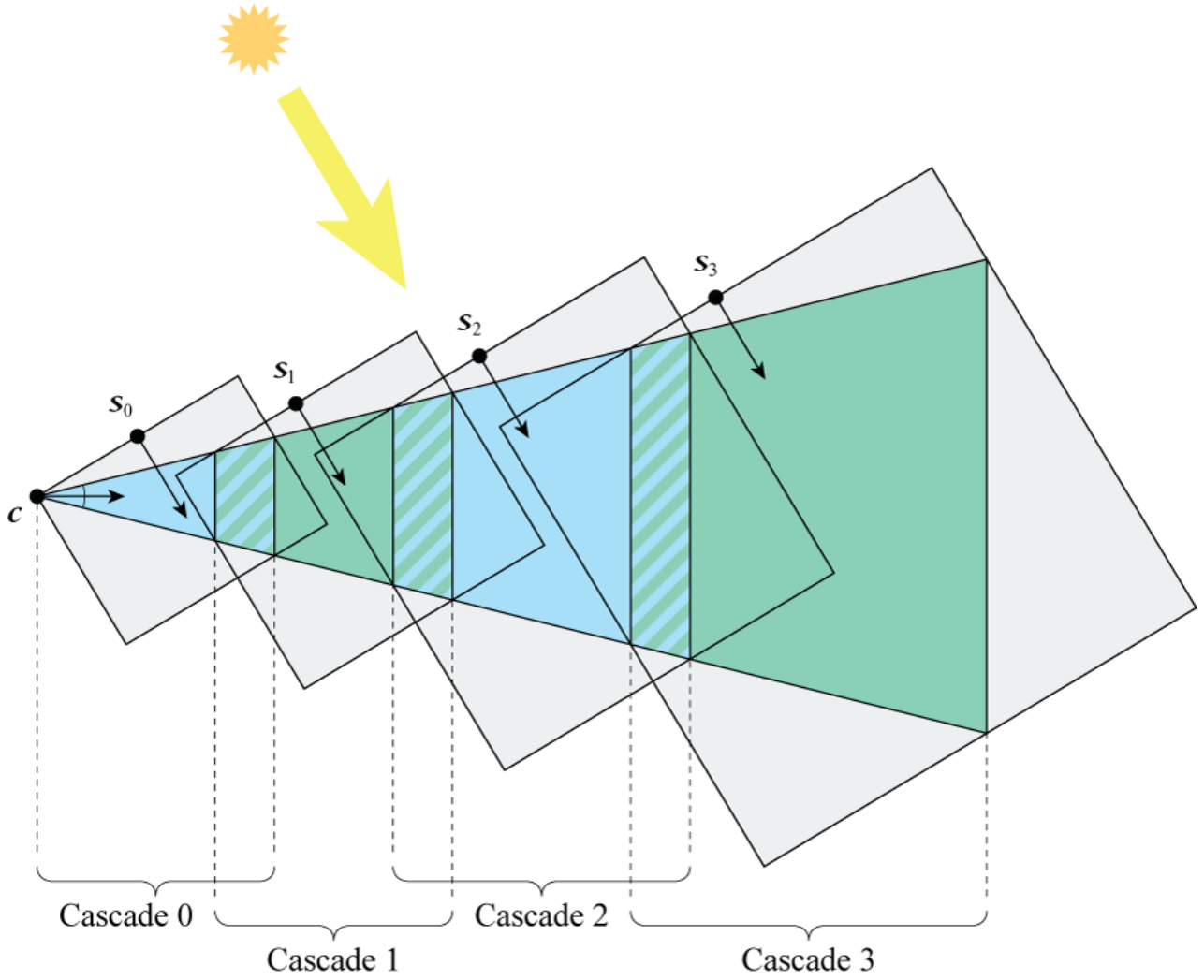
Cube Shadow Maps

(a)

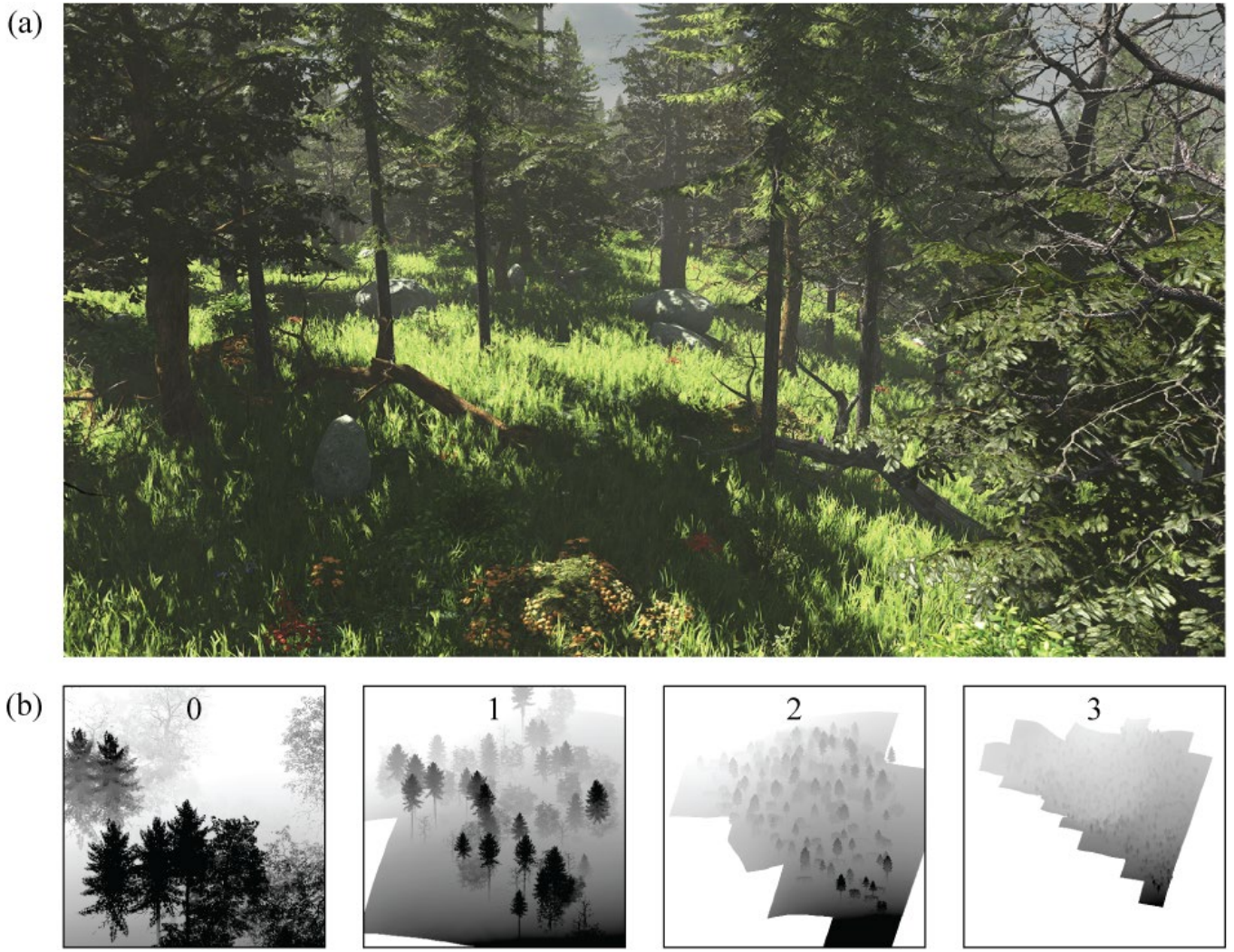
(b)



Cascaded Shadow Maps



Cascaded Shadow Maps



Thread Manager

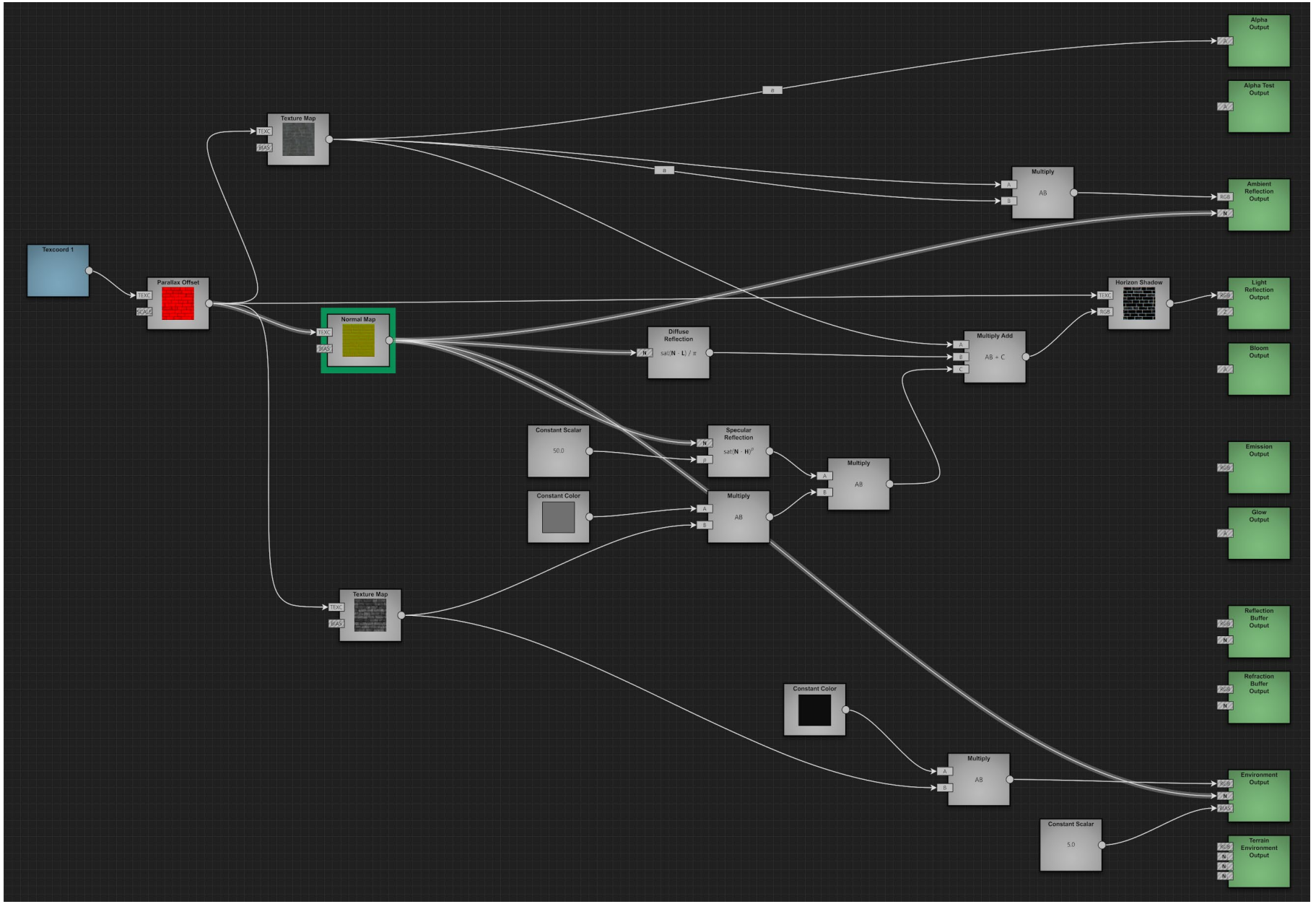
- Many processing cores available
- Divide per-frame processing into discrete jobs
  - Particle systems
  - Collision detection
  - Rope and cloth simulation
  - Character skinning
  - Shadow cascades / faces
- Queue jobs on worker threads, one per core

Graphs

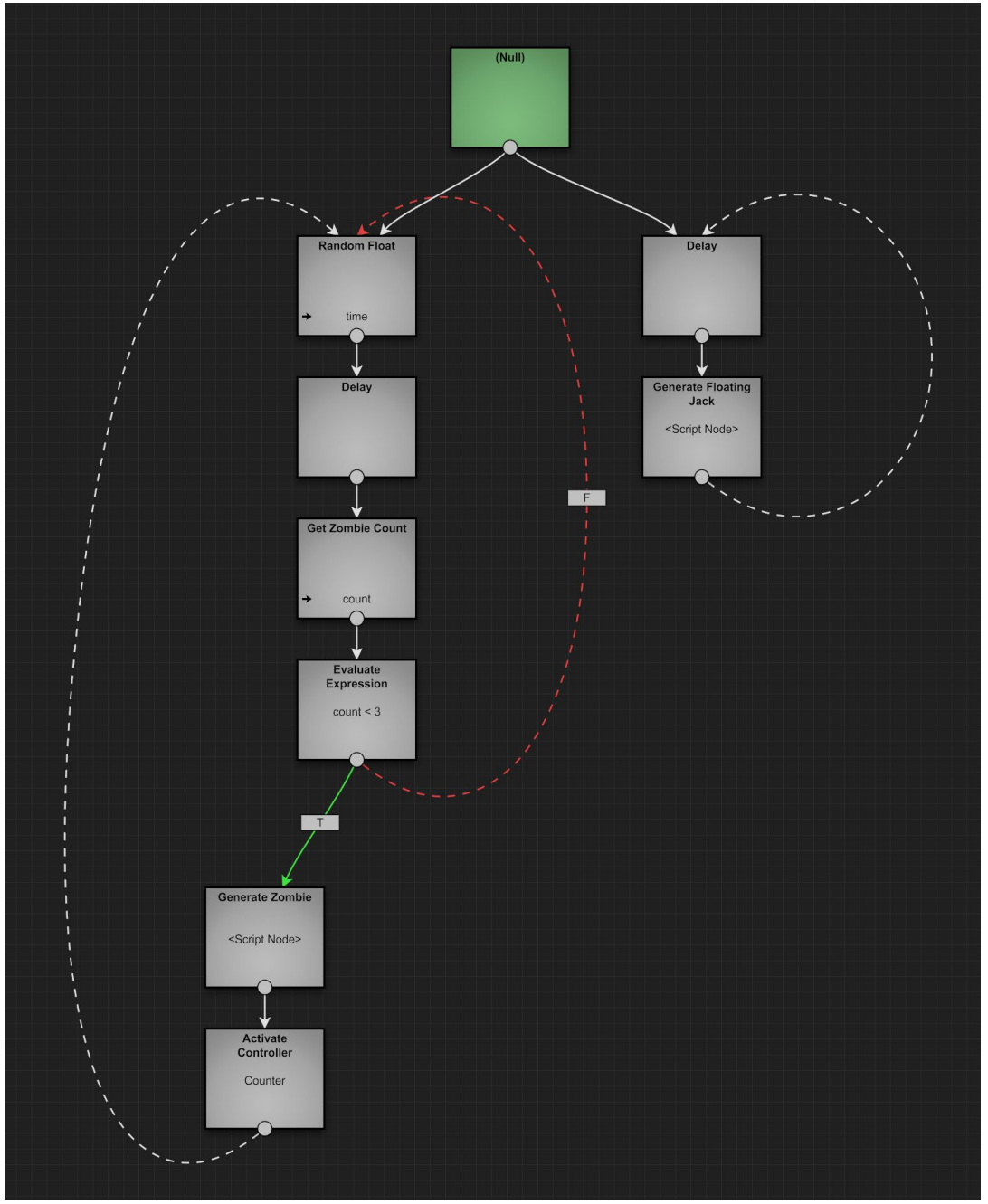
- Many graph structures used inside engine
  - Shaders (data flow)
  - Scripts (control flow)
  - Node connections
  - Physics contacts
  - Visibility graph

- A well-designed directed graph class is extremely handy

Shader Graph

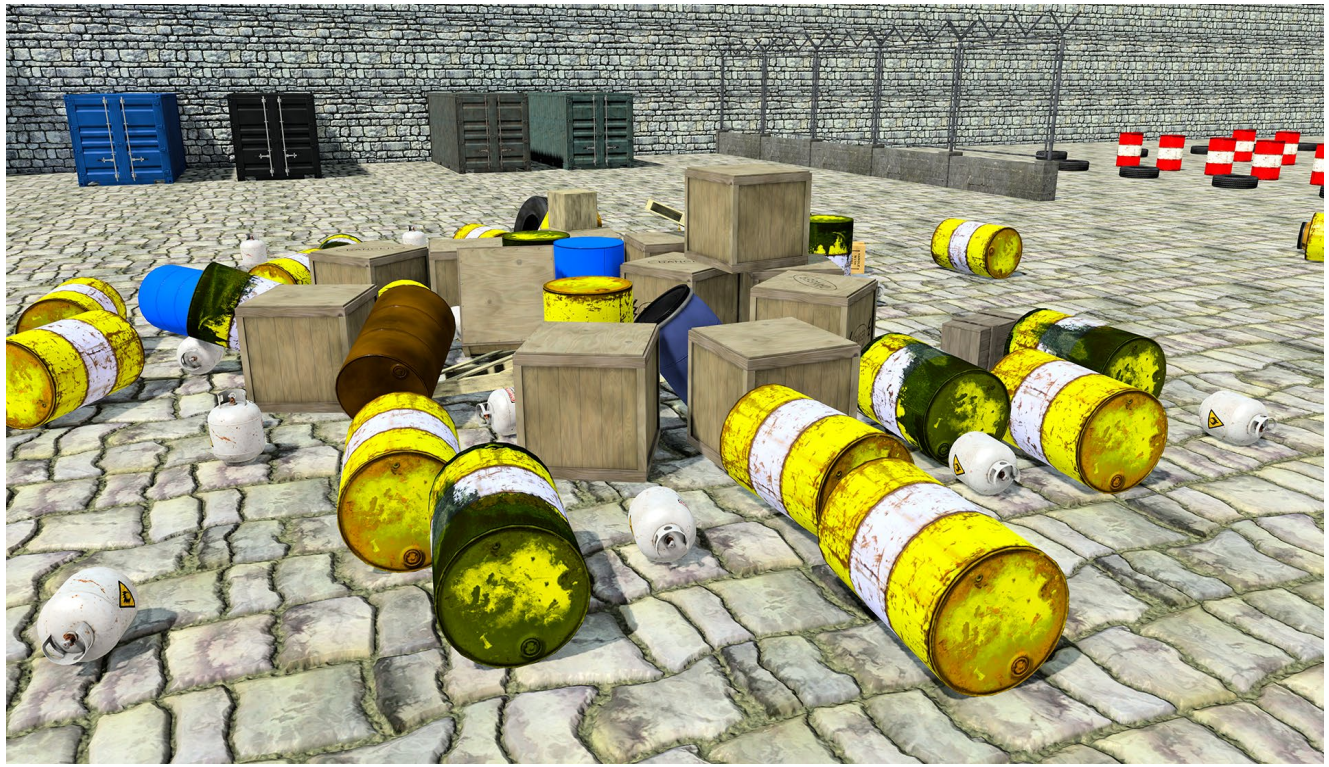


Script Graph



Physics

- Rigid bodies
- Breakable objects
- Character controllers
- Vehicle controllers
- Projectile controllers
- Water simulation
- Rope / cloth simulation



Contact

- lengyel@terathon.com
- Twitter: @EricLengyel
- Bluesky: @ericlengyel.bsky.social
- LinkedIn: www.linkedin.com/in/eric-lengyel