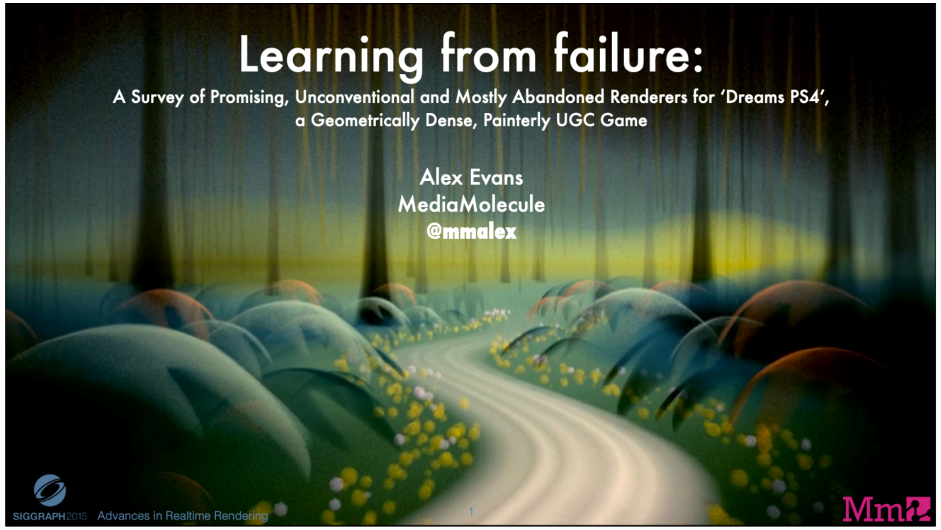
CLOSED-FORM IMPLICIT SURFACES ON THE GPU

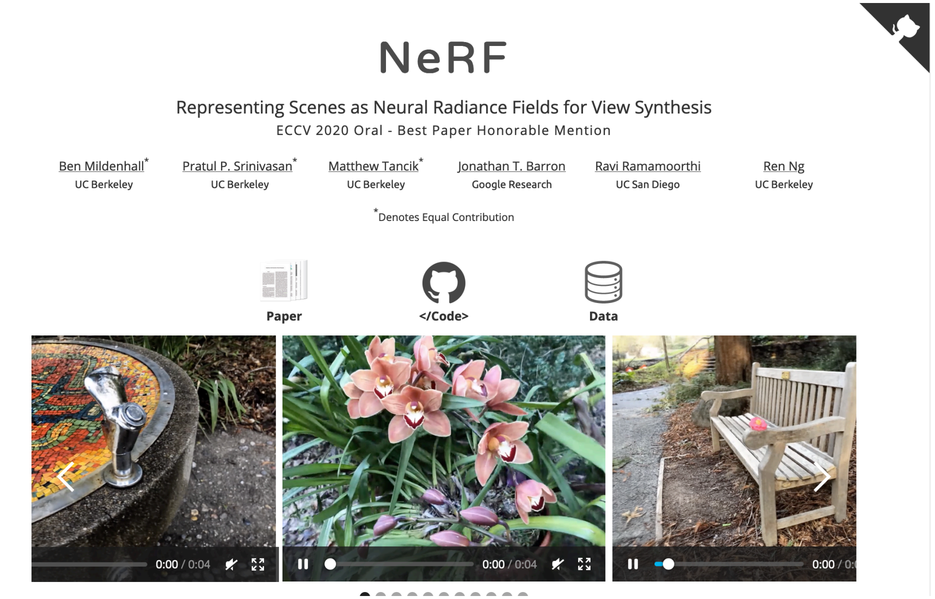
Matt Keeter
Independent researcher

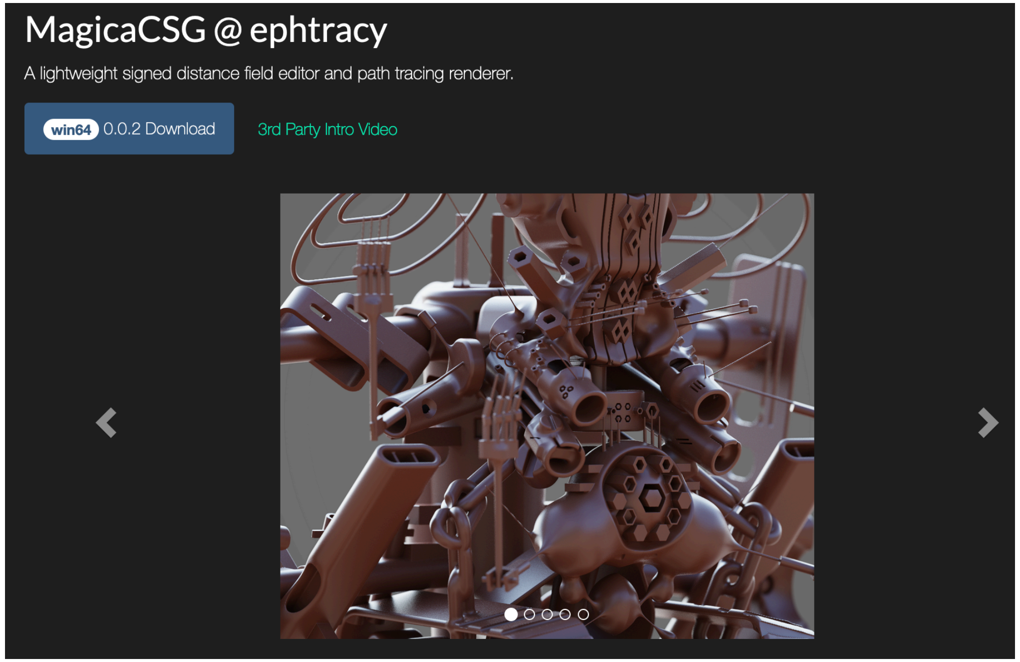
SHAPE REPRESENTATIONS

Boundary	Volumetric	
Triangle / quad meshes	Voxels	CSG
NURBS surfaces	Tet meshes	Implicit surfaces

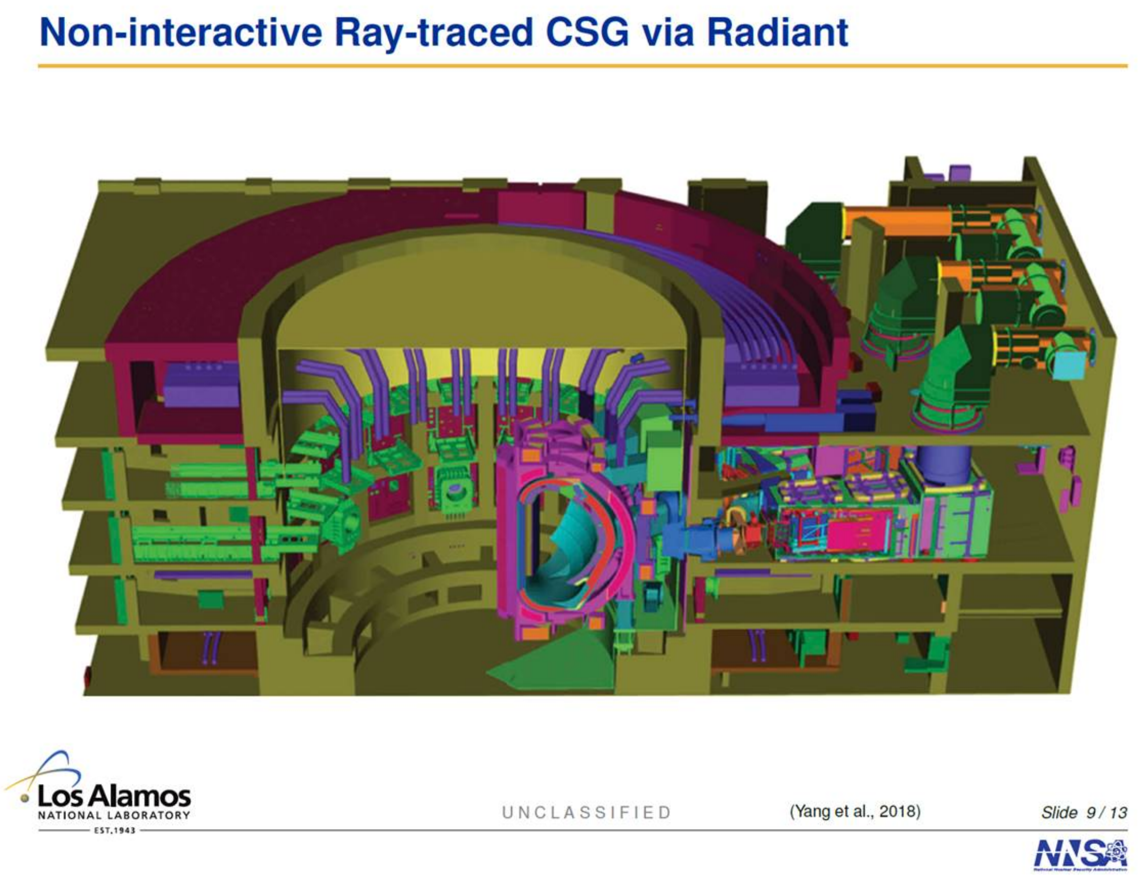
VOLUMETRIC RENAISSANCE

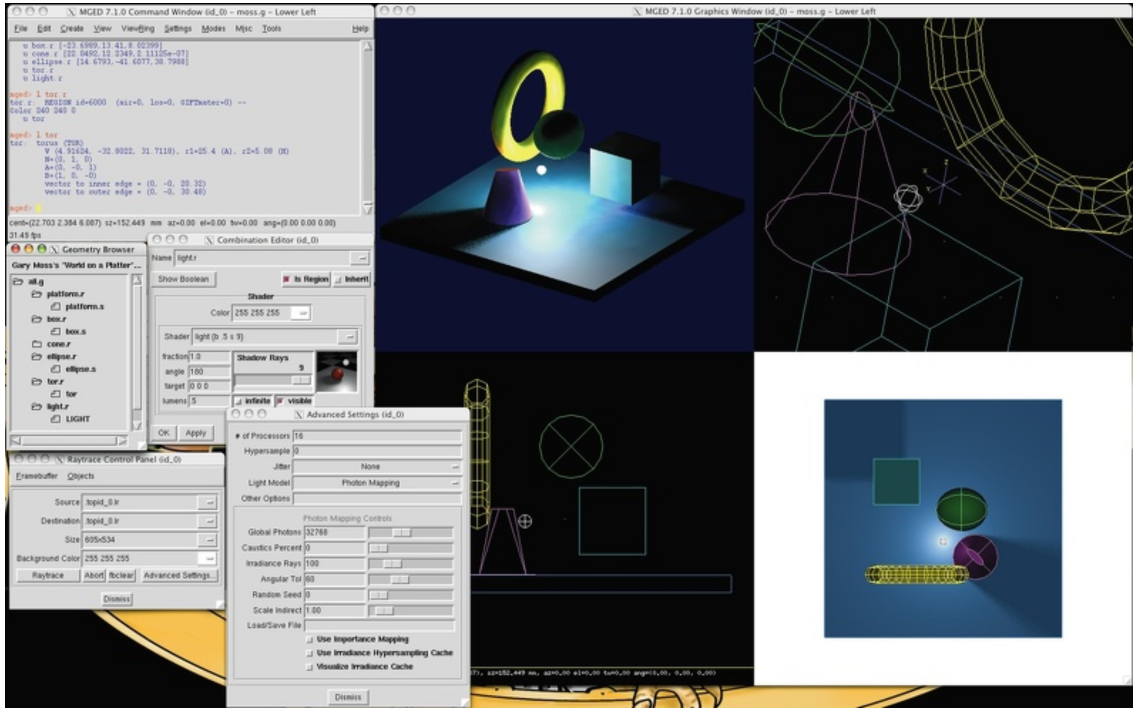






HISTORICAL VOLUMETRIC MODELING





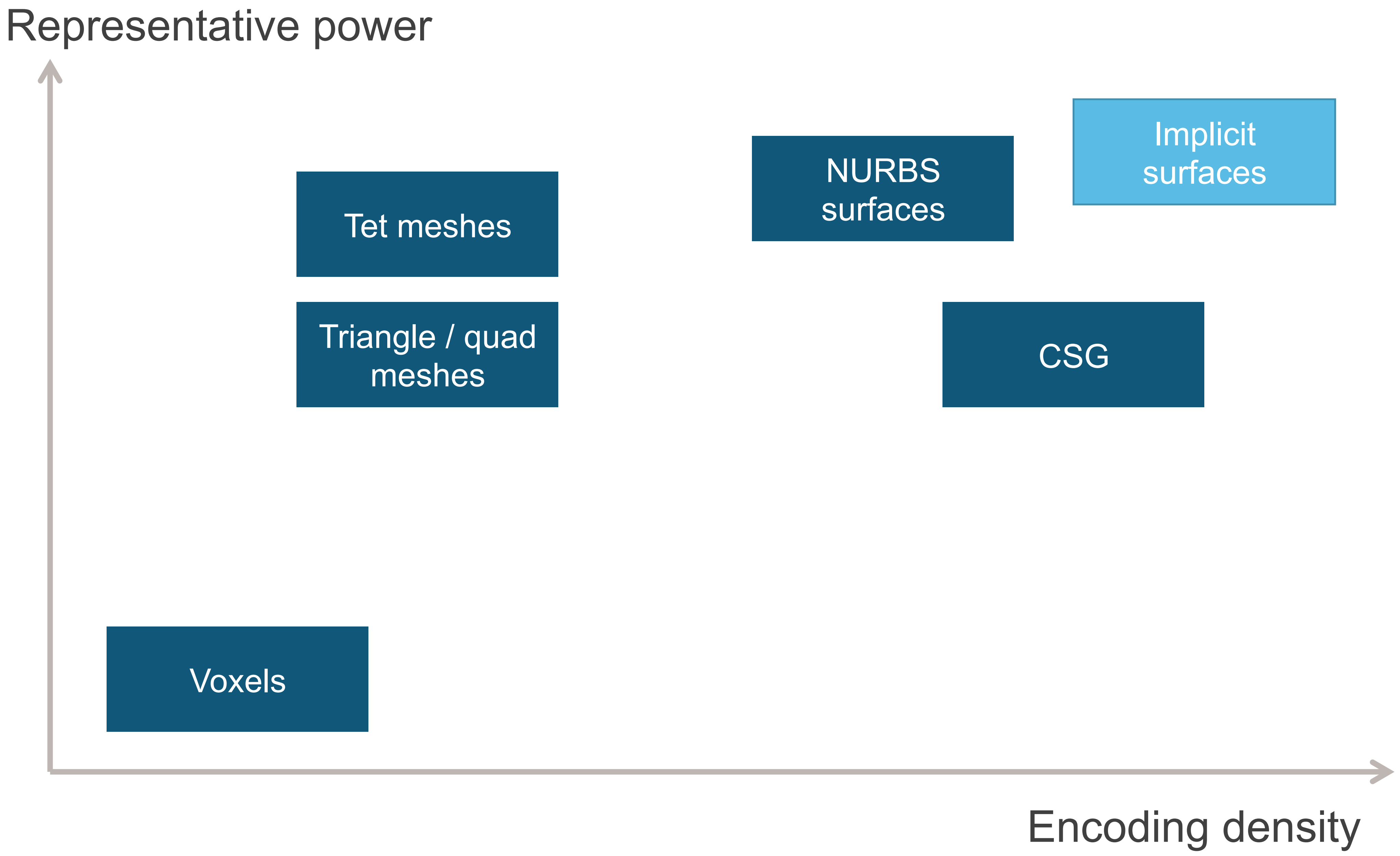
SHAPE REPRESENTATIONS

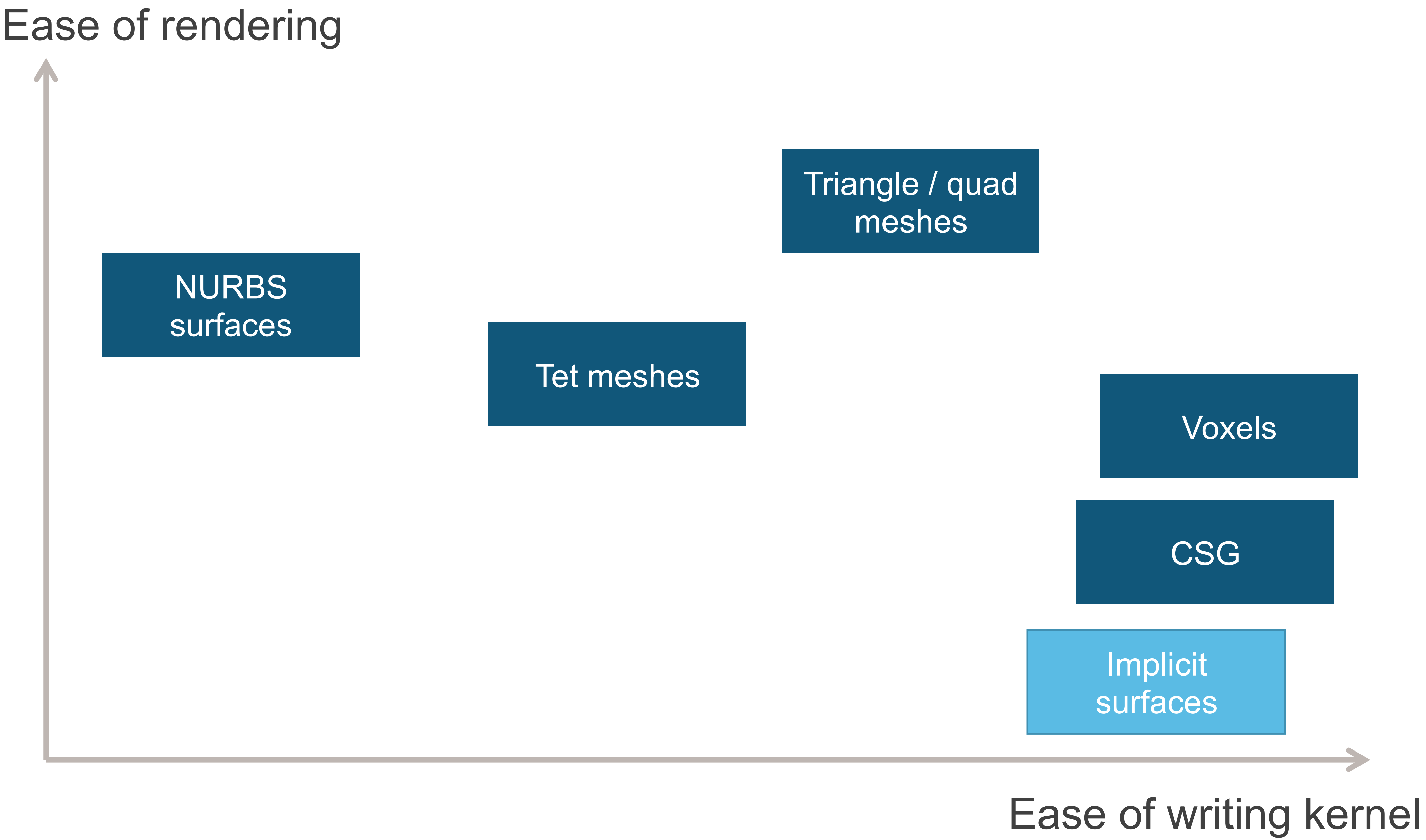
Boundary

- Triangle / quad meshes
- NURBS surfaces

Volumetric

- Voxels
- CSG
- Tet meshes
- Implicit surfaces


Representative power
Tet meshes
NURBS surfaces
Implicit surfaces
Triangle / quad meshes
CSG
Voxels
Encoding density


Ease of rendering
Triangle / quad meshes
NURBS surfaces
Tet meshes
Voxels
CSG
Implicit surfaces
Ease of writing kernel

Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces

MATTHEW J. KEETER, Independent researcher

Fig. 1. An assortment of implicit surfaces rendered using our technique. Left: an extruded text string, rotated and rendered as a heightmap. Center: a bear head sculpted using smooth blending operations, with normals found by automatic differentiation. Right: a complex architectural model rendered with screen-space ambient occlusion and perspective. All models are rendered directly from their mathematical representations, without triangulation or raytracing.

We present a new method for directly rendering complex closed-form implicit surfaces on modern GPUs, taking advantage of their massive parallelism. Our model representation is unambiguously solid, can be sampled at arbitrary resolution, and supports both constructive solid geometry (CSG) and more unusual modeling operations (e.g. smooth blending of shapes). The rendering strategy scales to large-scale models with thousands of arithmetic operations in their underlying mathematical expressions. Our method only requires C^0 continuity, allowing for warping and blending operations which break Lipshitz continuity.

To render a model, its underlying expression is evaluated in a shallow hierarchy of spatial regions, using a high branching factor for efficient parallelization. Interval arithmetic is used to both skip empty regions and construct reduced versions of the expression. The latter is the optimization that makes our algorithm practical: in one benchmark, expression complexity decreases by two orders of magnitude between the original and reduced expressions. Similar algorithms exist in the literature, but tend to be deeply recursive with heterogeneous workloads in each branch, which makes them GPU-unfriendly; our evaluation and expression reduction both run efficiently as massively parallel algorithms, entirely on the GPU.

The resulting system renders complex implicit surfaces in high resolution and at interactive speeds. We examine how performance scales with computing power, presenting performance results on hardware ranging from older laptops to modern data-center GPUs, and showing significant improvements at each stage.

CCS Concepts: • **Computing methodologies → Rasterization**; **Volumetric models**.

Additional Key Words and Phrases: implicit surface, signed distance field, freps, octrees, rasterization, gpu, cuda

Author's address: Matthew J. Keeter, Independent researcher, matt.j.keeter@gmail.com.

0730-0301/2020/7-ART141 $15.00
https://doi.org/10.1145/3386569.3392429

ACM Reference Format:
Matthew J. Keeter. 2020. Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces. *ACM Trans. Graph.* 39, 4, Article 141 (July 2020), 10 pages. https://doi.org/10.1145/3386569.3392429

1 INTRODUCTION

Implicit surfaces and functional representations are a powerful way to represent solid models [Bloomenthal and Wyvill 1997; Gomes et al. 2009]. Compared to boundary representations (e.g. triangle meshes or NURBS surfaces), they offer unambiguous inside-outside checking, easy constructive solid geometry (CSG) operations, and arbitrary resolution. In recent years, functional representations (f-reps) have been used as the kernel of both commercial [Courter 2019] and open-source [Keeter 2019] CAD packages. They are a fundamental building block in the demoscene community [Burger et al. 2002; Quilez 2008], used as a representation for generative art [Moen 2019], and even as the underlying technology for a recent PlayStation 4 game [Evans 2015].

Unlike boundary representations, implicit surfaces cannot easily be rendered in their native forms. This paper presents a new method for rendering the family of implicit surfaces represented by arbitrary closed-form arithmetic expressions, i.e., representing a sphere as

$$f(x,y,z) < 0 \text{ where } f(x,y,z) = \sqrt{x^2 + y^2 + z^2} - 1$$

This representation is particularly flexible and can be treated as an "assembly language for shapes" which is targeted by higher-level tools. The space of higher-level tools spans the gamut from advanced solid modeling packages [Allen 2019] to user-friendly content generation tools [Keeter 2015].

Our rendering strategy runs in both 2D and 3D, making efficient use of modern GPU hardware and APIs. Unlike previous work, it scales to complex expressions, maintaining interactive framerates while rendering models built from hundreds or thousands of arithmetic operations. It requires no continuity higher than C^0, which allows for extremely flexible modeling and unusual spatial transformations. Finally, it scales well with GPU power; as GPU performance

DEFINING IMPLICIT SURFACES

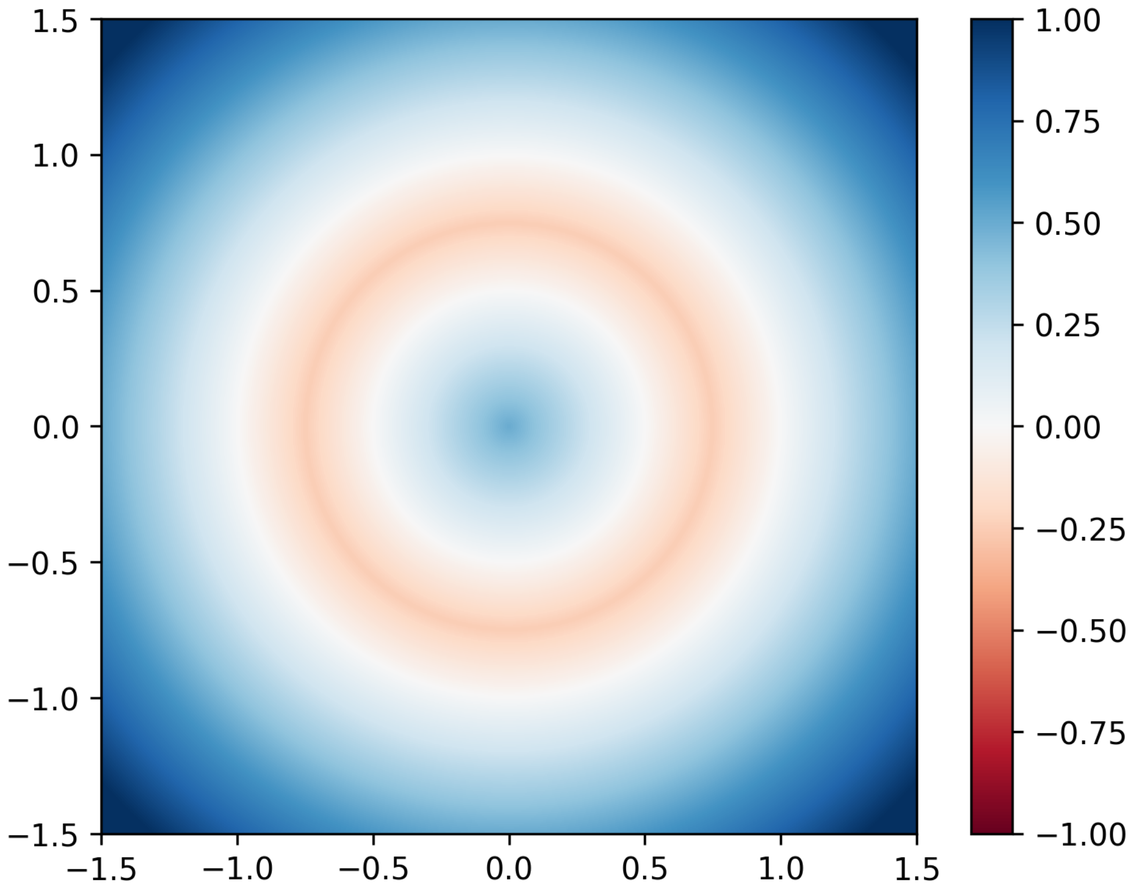
`f(x, y, z) < 0` → inside the shape ("filled")

`f(x, y, z) > 0` → outside the shape ("empty")

`f(x, y, z) = 0` → surface of the shape

CLOSED-FORM IMPLICIT SURFACES

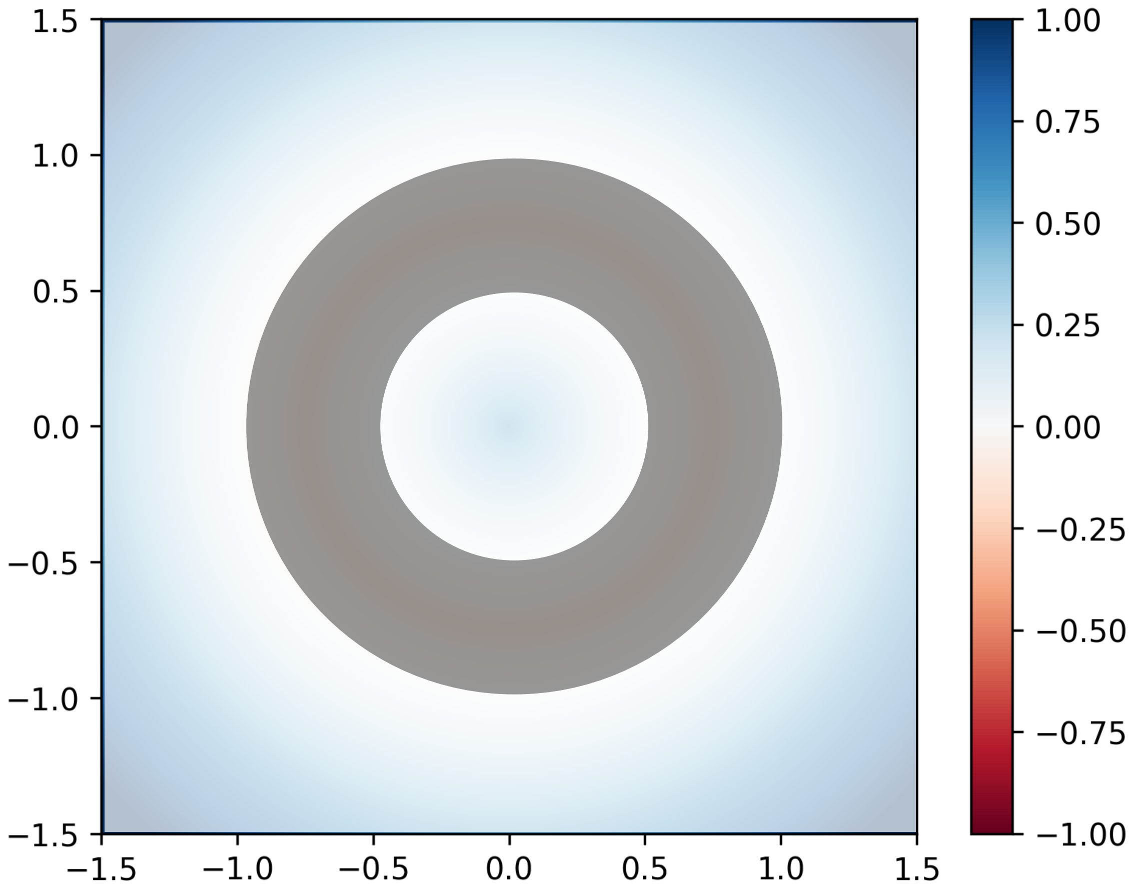
- `f(x, y, z)` is some sequence of mathematical operations
- Our reference implementation supports `+, -, ×, /, sqrt, sin, cos, asin, acos, atan, exp, log, abs, min, max`
- CSG is easy:
  - Intersection → `max`
  - Union → `min`
  - Inverse → `negation`



$$\max\left(0.5 - \sqrt{x^2 + y^2}, \sqrt{x^2 + y^2} - 1\right) < 0$$

<u>CLOSED-FORM</u> IMPLICIT SURFACES

- `f(x, y, z)` is some sequence of mathematical operations
- Our reference implementation supports `+`, `−`, `×`, `/`, `sqrt`, `sin`, `cos`, `asin`, `acos`, `atan`, `exp`, `log`, `abs`, `min`, `max`
- CSG is easy:
  - Intersection → `max`
  - Union → `min`
  - Inverse → `negation`



$$\max\left(0.5 - \sqrt{x^2 + y^2}, \sqrt{x^2 + y^2} - 1\right) < 0$$

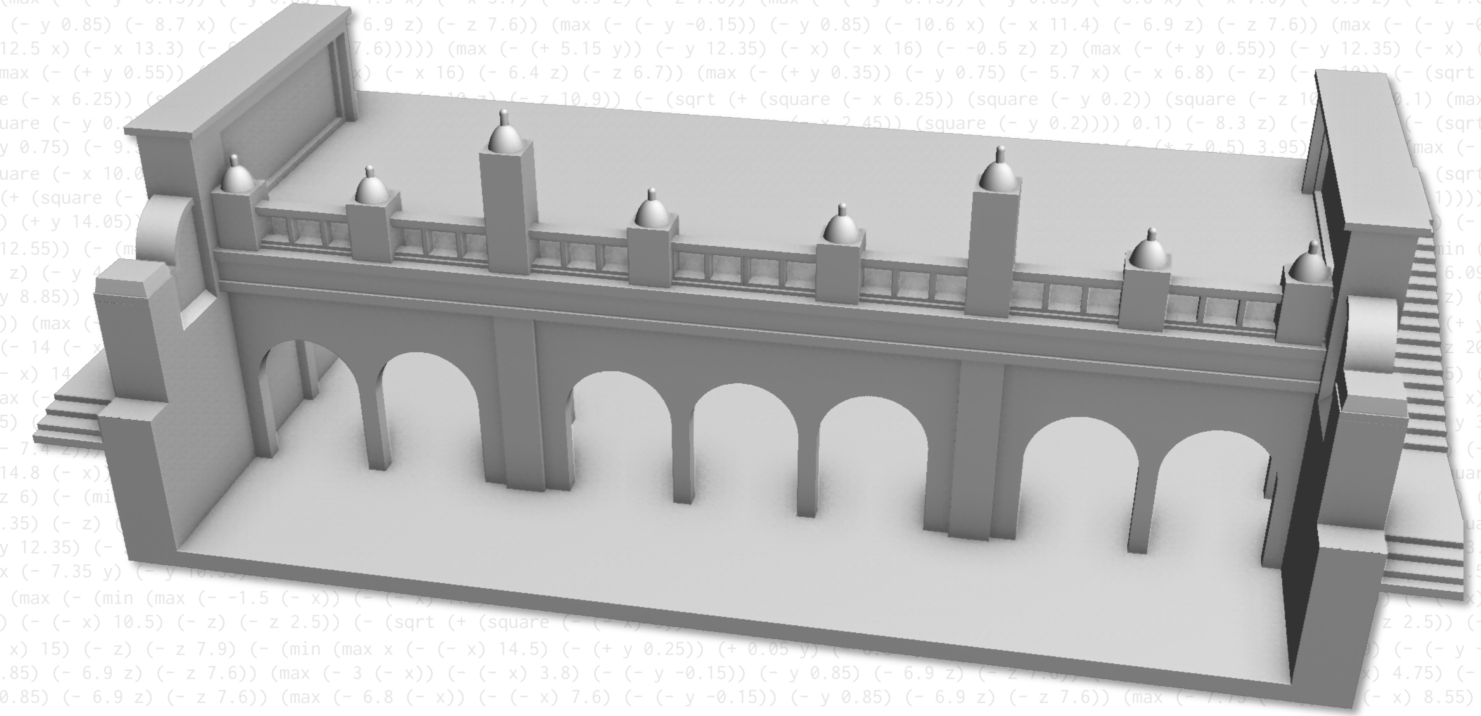
COMPLEX CLOSED-FORM IMPLICIT SURFACES

COMPLEX CLOSED-FORM IMPLICIT SURFACES

```
(min (max (- (+ 5.15 y)) (- y 12.35) (- 14.5 x) (- x 16) (- z) (- z 10) (- (min (max (- (+ 10.15 y)) (+ 0.75 y) (- 14 x) (- x 18) (- 7 z) (- z 20)) (max (- (+ 5.15 y)) (+ y 4.15) (- 14 x) (- x 18) (- 4 z) (- z 20)) (max (- 12 x) (- x 18) (-
9.85 y) (- y 13.85) (- 6.6 z) (- z 10)) (max (- (+ 2.75 y)) (+ 0.75 y) (- 14 x) (- x 14.8) (- 5 z) (- z 20) (- (- (+ z 6.2) (* x 0.8)))) (max (- 14 x) (- x 14.8) (- 0.85 y) (- y 9.55) (- 6.6 z) (- z 9.5)) (max (- 14 x) (- x 15) (- 1.05 y) (-
y 9.35) (- 6.8 z) (- z 9.3)) (max (- (+ 2.75 y)) (+ 0.75 y) (- x 16.5) (- 15.7 x) (- 5 z) (- z 20) (- (- (+ (* x 0.8) z) 18.2))) (max (- x 16.5) (- 15.7 x) (- 0.85 y) (- y 9.55) (- 6.6 z) (- z 9.5)) (max (- x 16.5) (- 15.5 x) (- 1.05 y) (- y
9.35) (- 6.8 z) (- z 9.3))))) (max (- (+ (* (- x 15.25) (/ 4 (- 10.8 z))) 0.75)) (- (* (- x 15.25) (/ 4 (- 10.8 z))) 0.75) (- (+ (* (+ y 3.45) (/ 4 (- 10.8 z))) 0.7)) (- (* (+ y 3.45) (/ 4 (- 10.8 z))) 0.7) (- (- z 6.8)) (- z 10.8) (- (- 7.4
z)) [illegible] .75 y)))) 1.5) (- 14.8 x) (- x 15.7)) (max
(- [illegible] z 6) (- (min (max (- 0.35 y) (- y 3.35) (-
z) (- z 2.5)) (- (sqrt (+ (square (- y 1.85)) (square (- z 2.5)))) 1.5) (max (- 3.85 y) (- y 6.85) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 5.35)) (square (- z 2.5)))) 1.5) (max (- 7.35 y) (- y 10.35) (- z) (- z 2.5)) (- (sqrt (+ (square (-
y 8.85)) (square (- z 2.5)))) 1.5) (max (- 10.85 y) (- y 13.85) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 12.35)) (square (- z 2.5)))) 1.5))) (- (- x 6)) (- x 6.5)) (max (- (- y -0.15)) (- y 12.35) (- z) (- z 6) (- (min (max (- 0.35 y) (- y
3.35) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 1.85)) (square (- z 2.5)))) 1.5) (max (- 3.85 y) (- y 6.85) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 5.35)) (square (- z 2.5)))) 1.5) (max (- 7.35 y) (- y 10.35) (- z) (- z 2.5)) (- (sqrt (+
(square (- y 8.85)) (square (- z 2.5)))) 1.5) (max (- 10.85 y) (- y 13.85) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 12.35)) (square (- z 2.5)))) 1.5))) (- (- x 14.2)) (- x 14.7)) (max (- (min (max (- -1.5 x) (- x 1.5) (- z) (- z 2.5)) (-
(sqrt (+ (square x) (square (- z 2.5)))) 1.5) (max (- 2 x) (- x 5) (- z) (- z 2.5)) (- (sqrt (+ (square (- x 3.5)) (square (- z 2.5)))) 1.5) (max (- 7.5 x) (- x 10.5) (- z) (- z 2.5)) (- (sqrt (+ (square (- x 9)) (square (- z 2.5)))) 1.5)
(max (- 11 x) (- x 14) (- z) (- z 2.5)) (- (sqrt (+ (square (- x 12.5)) (square (- z 2.5)))) 1.5))) (- (- y -0.15)) (- y 0.35) (- x) (- x 15) (- z) (- z 7.9) (- (min (max (- (+ y 0.25)) (+ 0.05 y) (- x) (- x 14.5) (- 6.8 z) (- z 7.7)) (max
(- (- y -0.15)) (- y 0.85) (- 0.1 x) (- x 0.9) (- 6.9 z) (- z 7.6)) (max (- (- y -0.15)) (- y 0.85) (- 1.05 x) (- x 1.85) (- 6.9 z) (- z 7.6)) (max (- (- y -0.15)) (- y 0.85) (- 3 x) (- x 3.8) (- 6.9 z) (- z 7.6)) (max (- (- y -0.15)) (- y
0.85) (- 3.95 x) (- x 4.75) (- 6.9 z) (- z 7.6)) (max (- (- y -0.15)) (- y 0.85) (- 4.9 x) (- x 5.7) (- 6.9 z) (- z 7.6)) (max (- (- y -0.15)) (- y 0.85) (- 6.8 x) (- x 7.6) (- 6.9 z) (- z 7.6)) (max (- (- y -0.15)) (- y 0.85) (- 7.75 x) (-
x 8.55) (- 6.9 z) (- z 7.6)) (max (- (- y -0.15)) (- y 0.85) (- 8.7 x) (- x 9.5) (- 6.9 z) (- z 7.6)) (max (- (- y -0.15)) (- y 0.85) (- 10.6 x) (- x 11.4) (- 6.9 z) (- z 7.6)) (max (- (- y -0.15)) (- y 0.85) (- 11.55 x) (- x 12.35) (- 6.9
z) (- z 7.6)) (max (- (- y -0.15)) (- y 0.85) (- 12.5 x) (- x 13.3) (- 6.9 z) (- z 7.6))))) (max (- (+ 5.15 y)) (- y 12.35) (- x) (- x 16) (- -0.5 z) z) (max (- (+ y 0.55)) (- y 12.35) (- x) (- x 16) (- 5.5 z) (- z 5.8)) (max (- (+ y 0.45))
(- y 12.35) (- x) (- x 16) (- 5.8 z) (- z 6.6)) (max (- (+ y 0.55)) (- y -0.15) (- x) (- x 16) (- 6.4 z) (- z 6.7)) (max (- (+ y 0.35)) (- y 0.75) (- 5.7 x) (- x 6.8) (- z) (- z 10)) (- (sqrt (+ (square (- x 6.25)) (square (- y 0.2)) (square
(- (* z 0.5) 4.8)))) 0.5) (max (- (sqrt (+ (square (- x 6.25)) (square (- y 0.2)))) 0.1) (- 10 z) (- z 10.9)) (- (sqrt (+ (square (- x 6.25)) (square (- y 0.2)) (square (- z 10.9)))) 0.1) (max (- (+ y 0.35)) (- y 0.75) (- 1.9 x) (- x 3) (- 6
z) (- z 8.3)) (- (sqrt (+ (square (- x 2.45)) (square (- y 0.2)) (square (- (* z 0.5) 3.95)))) 0.5) (max (- (sqrt (+ (square (- x 2.45)) (square (- y 0.2)))) 0.1) (- 8.3 z) (- z 9.1)) (- (sqrt (+ (square (- x 2.45)) (square (- y 0.2))
(square (- z 9.1)))) 0.1) (max (- (+ y 0.35)) (- y 0.75) (- 9.5 x) (- x 10.6) (- 6 z) (- z 8.3)) (- (sqrt (+ (square (- x 10.05)) (square (- y 0.2)) (square (- (* z 0.5) 3.95)))) 0.5) (max (- (sqrt (+ (square (- x 10.05)) (square (- y
0.2)))) 0.1) (- 8.3 z) (- z 9.1)) (- (sqrt (+ (square (- x 10.05)) (square (- y 0.2)) (square (- z 9.1)))) 0.1) (max (- (+ y 0.35)) (- y 0.75) (- 13.3 x) (- x 14.4) (- 6 z) (- z 8.3)) (- (sqrt (+ (square (- x 13.85)) (square (- y 0.2))
(square (- (* z 0.5) 3.95)))) 0.5) (max (- (sqrt (+ (square (- x 13.85)) (square (- y 0.2)))) 0.1) (- 8.3 z) (- z 9.1)) (- (sqrt (+ (square (- x 13.85)) (square (- y 0.2)) (square (- z 9.1)))) 0.1) (max (min (max (- (* z 3) (+ y 16.45)) (-
(min (- x 14.5) (- 18 x))) (- (min (max (- (* z 3) (+ y 14.05)) (- (min (- x 14.5) (- 18 x)))) (- 3.7 z) (+ y 5.15) (+ y 5.15)))) (max (- (+ 3.85 y)) (- y 0.15) (- 16 x) (- x 18) (- 3.4 z) (- z 3.7)) (max (- x) (- x 18) (- 8.75 y) (- y
12.35) (- 6.3 z) (- z 6.6)) (max (- (* z 3) (+ y 12.55)) (- (min (- x 14.5) (- 18 x))) (- (max (- (* z 3) (+ y 10.15)) (- (min (- x 14.5) (- 18 x))))) (- (min (- z 3.4) (- 6.6 z))))) (- (min (max (- 4.726 z) (- y 3.35)) (max (- 4.897 z) (- y
3.85)) (max (- 5.068 z) (- y 4.35)) (max (- 5.239 z) (- y 4.85)) (max (- 5.41 z) (- y 5.35)) (max (- 5.581 z) (- y 5.85)) (max (- 5.752 z) (- y 6.35)) (max (- 5.923 z) (- y 6.85)) (max (- 6.094 z) (- y 7.35)) (max (- 6.265 z) (- y 7.85))
(max (- 6.436 z) (- y 8.35)) (max (- 6.607 z) (- y 8.85)) (max (- 6.778 z) (- y 9.35)) (max (- 6.949 z) (- y 9.85)) (max (- 4.384 z) (- y 2.35)) (max (- 4.555 z) (- y 2.85)) (max (- 4.042 z) (- y 1.35)) (max (- 4.213 z) (- y 1.85)) (max (-
(- z 3.7)) (- y 0.35)) (max (- 3.871 z) (- y 0.85)) (max (- (- z 2.85)) (+ y 5.85)) (max (- 3.021 z) (+ y 5.35)) (max (- 3.192 z) (+ y 4.85)) (max (- 3.363 z) (+ y 4.35)) (max (- 3.534 z) (+ y 3.85))))) (max (- 14.5 (- x)) (- (- x) 16) (- (+
5.15 y)) (- y 12.35) (- z) (- z 10) (- (min (max (- 14 (- x)) (- (- x) 18) (- (+ 10.15 y)) (+ 0.75 y) (- 7 z) (- z 20)) (max (- 14 (- x)) (- (- x) 18) (- (+ 5.15 y)) (+ y 4.15) (- 4 z) (- z 20)) (max (- 12 (- x)) (- (- x) 18) (- 9.85 y) (- y
13.85) (- 6.6 z) (- z 10)) (max (- 14 (- x)) (- (- x) 14.8) (- (+ 2.75 y)) (+ 0.75 y) (- 5 z) (- z 20) (- (- (+ z 6.2) (* (- x) 0.8)))) (max (- 14 (- x)) (- (- x) 14.8) (- 0.85 y) (- y 9.55) (- 6.6 z) (- z 9.5)) (max (- 14 (- x)) (- (- x)
15) (- 1.05 y) (- y 9.35) (- 6.8 z) (- z 9.3)) (max (- (- x) 16.5) (- 15.7 (- x)) (- (+ 2.75 y)) (+ 0.75 y) (- 5 z) (- z 20) (- (- (+ (* (- x) 0.8) z) 18.2))) (max (- (- x) 16.5) (- 15.7 (- x)) (- 0.85 y) (- y 9.55) (- 6.6 z) (- z 9.5)) (max
(- (- x) 16.5) (- 15.5 (- x)) (- 1.05 y) (- y 9.35) (- 6.8 z) (- z 9.3))))) (max (- (+ (* (- (- x) 15.25) (/ 4 (- 10.8 z))) 0.75)) (- (* (- (- x) 15.25) (/ 4 (- 10.8 z))) 0.75) (- (+ (* (+ y 3.45) (/ 4 (- 10.8 z))) 0.7)) (- (* (+ y 3.45) (/
4 (- 10.8 z))) 0.7) (- (- z 6.8)) (- z 10.8) (- (- 7.4 z))) (max (- 14.7 (- x)) (- (- x) 15.8) (- (+ 3.15 y)) (+ y 2.05) (- 6.8 z) (- z 7)) (max (- 14.3 (- x)) (- (- x) 16.2) (- (+ y 0.95)) (- y 9.85) (- 10 z) (- z 10.2)) (max (- (sqrt (+
(square (- z 6.8)) (square (+ 0.75 y)))) 1.5) (- 14.8 (- x)) (- (- x) 15.7)) (max (- (- (sqrt (+ (square (- z 6.8)) (square (+ 0.75 y)))) 1.3)) (- 14.7 (- x)) (- (- x) 15.8) (- (sqrt (+ (square (- z 6.8)) (square (+ 0.75 y)))) 1.5) (- (- z
6.8))) (max (- (- y -0.15)) (- y 12.35) (- z) (- z 6) (- (min (max (- 0.35 y) (- y 3.35) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 1.85)) (square (- z 2.5)))) 1.5) (max (- 3.85 y) (- y 6.85) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 5.35))
(square (- z 2.5)))) 1.5) (max (- 7.35 y) (- y 10.35) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 8.85)) (square (- z 2.5)))) 1.5) (max (- 10.85 y) (- y 13.85) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 12.35)) (square (- z 2.5)))) 1.5))) (- (-
(- x) 6)) (- (- x) 6.5)) (max (- (- y -0.15)) (- y 12.35) (- z) (- z 6) (- (min (max (- 0.35 y) (- y 3.35) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 1.85)) (square (- z 2.5)))) 1.5) (max (- 3.85 y) (- y 6.85) (- z) (- z 2.5)) (- (sqrt (+
(square (- y 5.35)) (square (- z 2.5)))) 1.5) (max (- 7.35 y) (- y 10.35) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 8.85)) (square (- z 2.5)))) 1.5) (max (- 10.85 y) (- y 13.85) (- z) (- z 2.5)) (- (sqrt (+ (square (- y 12.35)) (square (- z
2.5)))) 1.5))) (- (- (- x) 14.2)) (- (- x) 14.7)) (max (- (min (max (- -1.5 (- x)) (- (- x) 1.5) (- z) (- z 2.5)) (- (sqrt (+ (square (- x)) (square (- z 2.5)))) 1.5) (max (- 2 (- x)) (- (- x) 5) (- z) (- z 2.5)) (- (sqrt (+ (square (- (- x)
3.5)) (square (- z 2.5)))) 1.5) (max (- 7.5 (- x)) (- (- x) 10.5) (- z) (- z 2.5)) (- (sqrt (+ (square (- (- x) 9)) (square (- z 2.5)))) 1.5) (max (- 11 (- x)) (- (- x) 14) (- z) (- z 2.5)) (- (sqrt (+ (square (- (- x) 12.5)) (square (- z
2.5)))) 1.5))) (- (- y -0.15)) (- y 0.35) x (- (- x) 15) (- z) (- z 7.9) (- (min (max x (- (- x) 14.5) (- (+ y 0.25)) (+ 0.05 y) (- 6.8 z) (- z 7.7)) (max (- 0.1 (- x)) (- (- x) 0.9) (- (- y -0.15)) (- y 0.85) (- 6.9 z) (- z 7.6)) (max (-
1.05 (- x)) (- (- x) 1.85) (- (- y -0.15)) (- y 0.85) (- 6.9 z) (- z 7.6)) (max (- 3 (- x)) (- (- x) 3.8) (- (- y -0.15)) (- y 0.85) (- 6.9 z) (- z 7.6)) (max (- 3.95 (- x)) (- (- x) 4.75) (- (- y -0.15)) (- y 0.85) (- 6.9 z) (- z 7.6)) (max
(- 4.9 (- x)) (- (- x) 5.7) (- (- y -0.15)) (- y 0.85) (- 6.9 z) (- z 7.6)) (max (- 6.8 (- x)) (- (- x) 7.6) (- (- y -0.15)) (- y 0.85) (- 6.9 z) (- z 7.6)) (max (- 7.75 (- x)) (- (- x) 8.55) (- (- y -0.15)) (- y 0.85) (- 6.9 z) (- z 7.6))
(max (- 8.7 (- x)) (- (- x) 9.5) (- (- y -0.15)) (- y 0.85) (- 6.9 z) (- z 7.6)) (max (- 10.6 (- x)) (- (- x) 11.4) (- (- y -0.15)) (- y 0.85) (- 6.9 z) (- z 7.6)) (max (- 11.55 (- x)) (- (- x) 12.35) (- (- y -0.15)) (- y 0.85) (- 6.9 z) (-
z 7.6)) (max (- 12.5 (- x)) (- (- x) 13.3) (- (- y -0.15)) (- y 0.85) (- 6.9 z) (- z 7.6))))) (max x (- (- x) 16) (- (+ 5.15 y)) (- y 12.35) (- -0.5 z) z) (max x (- (- x) 16) (- (+ y 0.55)) (- y 12.35) (- 5.5 z) (- z 5.8)) (max x (- (- x)
16) (- (+ y 0.45)) (- y 12.35) (- 5.8 z) (- z 6.6)) (max x (- (- x) 16) (- (+ y 0.55)) (- y -0.15) (- 6.4 z) (- z 6.7)) (max (- 5.7 (- x)) (- (- x) 6.8) (- (+ y 0.35)) (- y 0.75) (- z) (- z 10)) (- (sqrt (+ (square (- (- x) 6.25)) (square (-
y 0.2)) (square (- (* z 0.5) 4.8)))) 0.5) (max (- (sqrt (+ (square (- (- x) 6.25)) (square (- y 0.2)))) 0.1) (- 10 z) (- z 10.9)) (- (sqrt (+ (square (- (- x) 6.25)) (square (- y 0.2)) (square (- z 10.9)))) 0.1) (max (- 1.9 (- x)) (- (- x)
3) (- (+ y 0.35)) (- y 0.75) (- 6 z) (- z 8.3)) (- (sqrt (+ (square (- (- x) 2.45)) (square (- y 0.2)) (square (- (* z 0.5) 3.95)))) 0.5) (max (- (sqrt (+ (square (- (- x) 2.45)) (square (- y 0.2)))) 0.1) (- 8.3 z) (- z 9.1)) (- (sqrt (+
(square (- (- x) 2.45)) (square (- y 0.2)) (square (- z 9.1)))) 0.1) (max (- 9.5 (- x)) (- (- x) 10.6) (- (+ y 0.35)) (- y 0.75) (- 6 z) (- z 8.3)) (- (sqrt (+ (square (- (- x) 10.05)) (square (- y 0.2)) (square (- (* z 0.5) 3.95)))) 0.5)
(max (- (sqrt (+ (square (- (- x) 10.05)) (square (- y 0.2)))) 0.1) (- 8.3 z) (- z 9.1)) (- (sqrt (+ (square (- (- x) 10.05)) (square (- y 0.2)) (square (- z 9.1)))) 0.1) (max (- 13.3 (- x)) (- (- x) 14.4) (- (+ y 0.35)) (- y 0.75) (- 6 z)
(- z 8.3)) (- (sqrt (+ (square (- (- x) 13.85)) (square (- y 0.2)) (square (- (* z 0.5) 3.95)))) 0.5) (max (- (sqrt (+ (square (- (- x) 13.85)) (square (- y 0.2)))) 0.1) (- 8.3 z) (- z 9.1)) (- (sqrt (+ (square (- (- x) 13.85)) (square (- y
0.2)) (square (- z 9.1)))) 0.1) (max (min (max (- (* z 3) (+ y 16.45)) (- (min (- (- x) 14.5) (- 18 (- x))))) (- (min (max (- (* z 3) (+ y 14.05)) (- (min (- (- x) 14.5) (- 18 (- x))))) (- 3.7 z) (+ y 5.15) (+ y 5.15)))) (max (- 16 (- x)) (-
(- x) 18) (- (+ 3.85 y)) (- y 0.15) (- 3.4 z) (- z 3.7)) (max x (- (- x) 18) (- 8.75 y) (- y 12.35) (- 6.3 z) (- z 6.6)) (max (- (* z 3) (+ y 12.55)) (- (min (- (- x) 14.5) (- 18 (- x)))) (- (max (- (* z 3) (+ y 10.15)) (- (min (- (- x)
14.5) (- 18 (- x)))))) (- (min (- z 3.4) (- 6.6 z))))) (- (min (max (- 4.726 z) (- y 3.35)) (max (- 4.897 z) (- y 3.85)) (max (- 5.068 z) (- y 4.35)) (max (- 5.239 z) (- y 4.85)) (max (- 5.41 z) (- y 5.35)) (max (- 5.581 z) (- y 5.85)) (max
(- 5.752 z) (- y 6.35)) (max (- 5.923 z) (- y 6.85)) (max (- 6.094 z) (- y 7.35)) (max (- 6.265 z) (- y 7.85)) (max (- 6.436 z) (- y 8.35)) (max (- 6.607 z) (- y 8.85)) (max (- 6.778 z) (- y 9.35)) (max (- 6.949 z) (- y 9.85)) (max (- 4.384
z) (- y 2.35)) (max (- 4.555 z) (- y 2.85)) (max (- 4.042 z) (- y 1.35)) (max (- 4.213 z) (- y 1.85)) (max (- (- z 3.7)) (- y 0.35)) (max (- 3.871 z) (- y 0.85)) (max (- (- z 2.85)) (+ y 5.85)) (max (- 3.021 z) (+ y 5.35)) (max (- 3.192 z)
(+ y 4.85)) (max (- 3.363 z) (+ y 4.35)) (max (- 3.534 z) (+ y 3.85))))))
```

COMPLEX CLOSED-FORM IMPLICIT SURFACES

```
(min (max (- (+ 5.15 y)) (- y 12.35) (- 14.5 x) (- x 16) (- z) (- z 10) (- (min (max (- (+ 10.15 y)) (+ 0.75 y) (- 14 x) (- x 18) (- 7 z) (- z 20)) (max (- (+ 5.15 y)) (+ y 4.15) (- 14 x) (- x 18) (- 4 z) (- z 20)) (max (- 12 x) (- x 18) (-
```



```
y 0.2)) (square (- (* z 0.5) 4.8)))) 0.5) (max (- (sqrt (+ (square (- (- x) 6.25)) (square (- y 0.2)))) 0.1) (- 10 z) (- z 10.9)) (- (sqrt (+ (square (- (- x) 6.25)) (square (- y 0.2)) (square (- z 10.9)))) 0.1) (max (- 1.9 (- x)) (- (- x)
3) (- (+ y 0.35)) (- y 0.75) (- 6 z) (- z 8.3)) (- (sqrt (+ (square (- (- x) 2.45)) (square (- y 0.2)) (square (- (* z 0.5) 3.95)))) 0.5) (max (- (sqrt (+ (square (- (- x) 2.45)) (square (- y 0.2)))) 0.1) (- 8.3 z) (- z 9.1)) (- (sqrt (+
(square (- (- x) 2.45)) (square (- y 0.2)) (square (- z 9.1)))) 0.1) (max (- 9.5 (- x)) (- (- x) 10.6) (- (+ y 0.35)) (- y 0.75) (- 6 z) (- z 8.3)) (- (sqrt (+ (square (- (- x) 10.05)) (square (- y 0.2)) (square (- (* z 0.5) 3.95)))) 0.5)
(max (- (sqrt (+ (square (- (- x) 10.05)) (square (- y 0.2)))) 0.1) (- 8.3 z) (- z 9.1)) (- (sqrt (+ (square (- (- x) 10.05)) (square (- y 0.2)) (square (- z 9.1)))) 0.1) (max (- 13.3 (- x)) (- (- x) 14.4) (- (+ y 0.35)) (- y 0.75) (- 6 z)
(- z 8.3)) (- (sqrt (+ (square (- (- x) 13.85)) (square (- y 0.2)) (square (- (* z 0.5) 3.95)))) 0.5) (max (- (sqrt (+ (square (- (- x) 13.85)) (square (- y 0.2)))) 0.1) (- 8.3 z) (- z 9.1)) (- (sqrt (+ (square (- (- x) 13.85)) (square (- y
0.2)) (square (- z 9.1)))) 0.1) (max (min (max (- (* z 3) (+ y 16.45)) (- (min (- (- x) 14.5) (- 18 (- x)))) (- (min (max (- (* z 3) (+ y 14.05)) (- (min (- (- x) 14.5) (- 18 (- x))))) (- 3.7 z) (+ y 5.15) (+ y 5.15)))) (max (- 16 (- x)) (-
(- x) 18) (- (+ 3.85 y)) (- y 0.15) (- 3.4 z) (- z 3.7)) (max x (- (- x) 18) (- 8.75 y) (- y 12.35) (- 6.3 z) (- z 6.6)) (max (- (* z 3) (+ y 12.55)) (- (min (- (- x) 14.5) (- 18 (- x)))) (- (max (- (* z 3) (+ y 10.15)) (- (min (- (- x)
14.5) (- 18 (- x)))))) (- (min (- z 3.4) (- 6.6 z))))) (- (min (max (- 4.726 z) (- y 3.35)) (max (- 4.897 z) (- y 3.85)) (max (- 5.068 z) (- y 4.35)) (max (- 5.239 z) (- y 4.85)) (max (- 5.41 z) (- y 5.35)) (max (- 5.581 z) (- y 5.85)) (max
(- 5.752 z) (- y 6.35)) (max (- 5.923 z) (- y 6.85)) (max (- 6.094 z) (- y 7.35)) (max (- 6.265 z) (- y 7.85)) (max (- 6.436 z) (- y 8.35)) (max (- 6.607 z) (- y 8.85)) (max (- 6.778 z) (- y 9.35)) (max (- 6.949 z) (- y 9.85)) (max (- 4.384
z) (- y 2.35)) (max (- 4.555 z) (- y 2.85)) (max (- 4.042 z) (- y 1.35)) (max (- 4.213 z) (- y 1.85)) (max (- (- z 3.7)) (- y 0.35)) (max (- 3.871 z) (- y 0.85)) (max (- (- z 2.85)) (+ y 5.85)) (max (- 3.021 z) (+ y 5.35)) (max (- 3.192 z)
(+ y 4.85)) (max (- 3.363 z) (+ y 4.35)) (max (- 3.534 z) (+ y 3.85))))))
```

“ASSEMBLY LANGUAGE FOR SHAPES”

Benefits

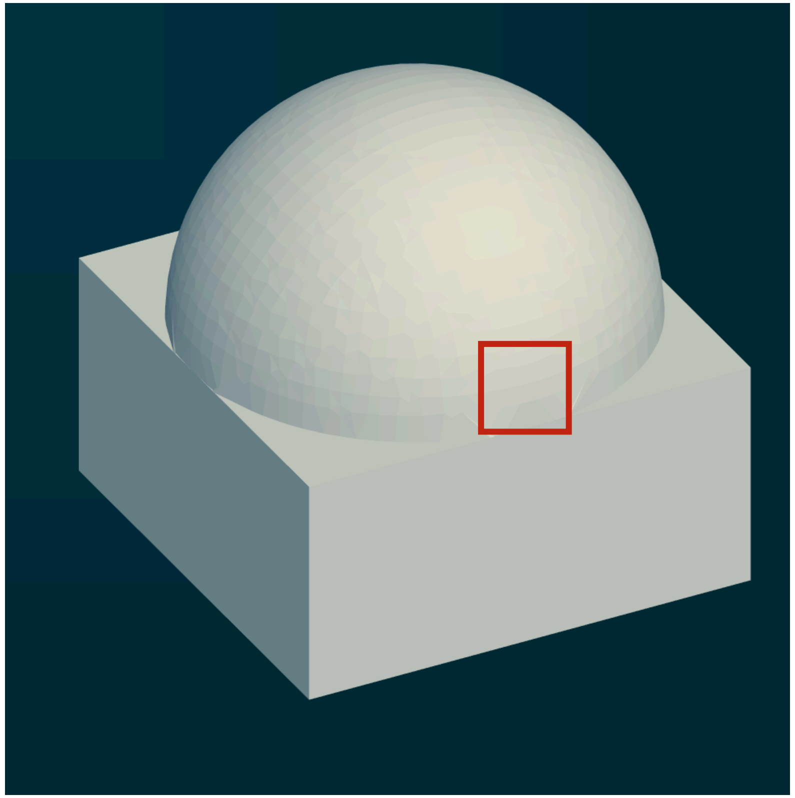
- Easy CSG and solid modeling!
- Unambiguous inside-outside checking
- Supports arbitrary resolution
- Compact representation

Applications

- CAD/CAM
- Demoscene graphics
- Art and exploration
- User-generated content

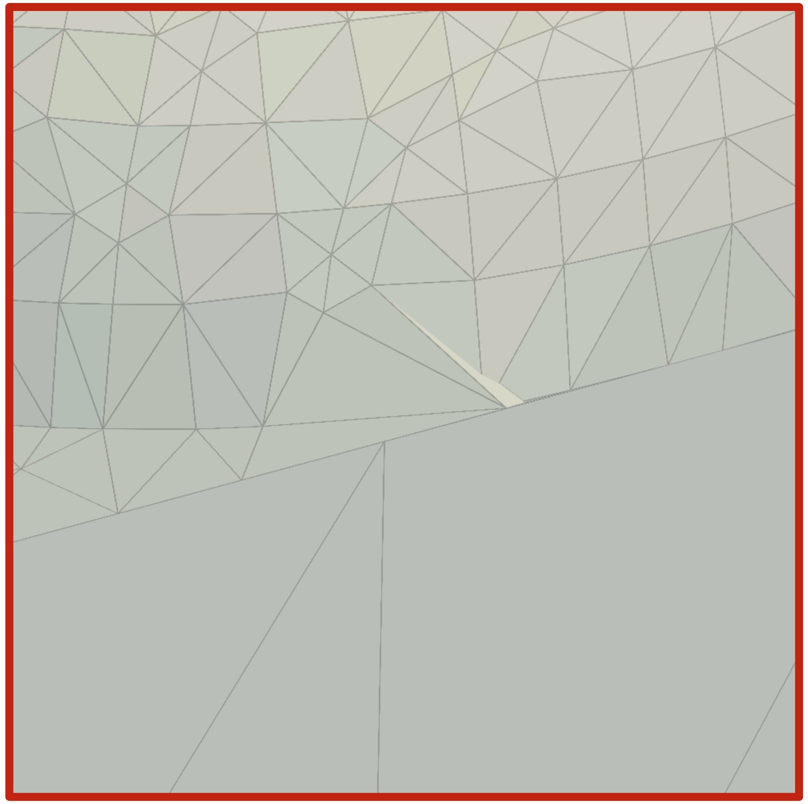
RENDERING IMPLICIT SURFACES IS HARD

- Meshing
  - Hard to get right for arbitrary shapes
  - Philosophically unsatisfying



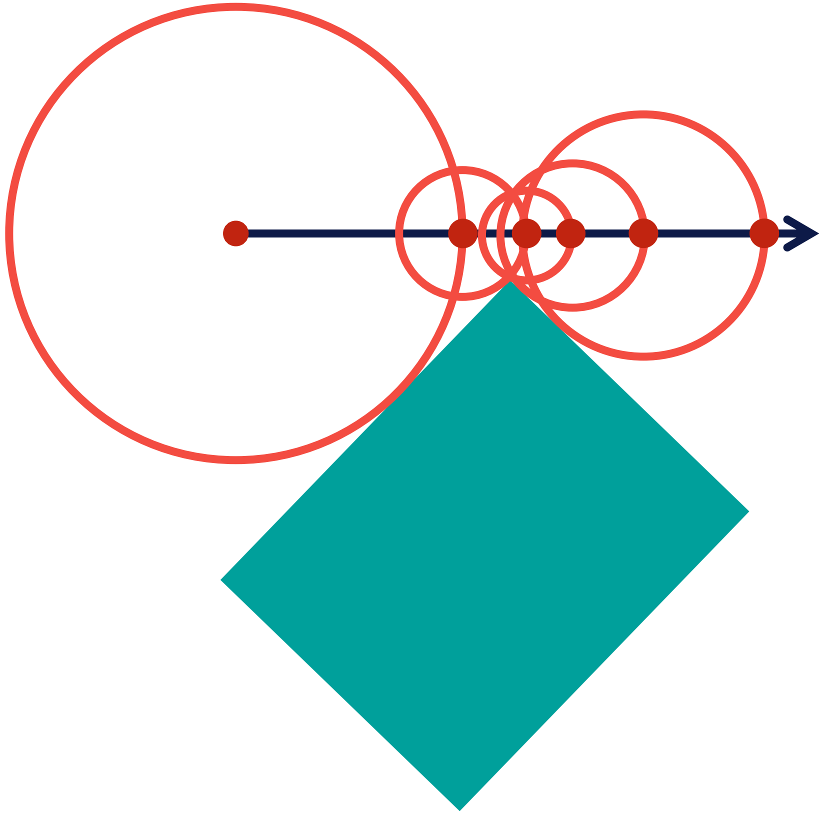
RENDERING IMPLICIT SURFACES IS HARD

- Meshing
  - Hard to get right for arbitrary shapes
  - Philosophically unsatisfying



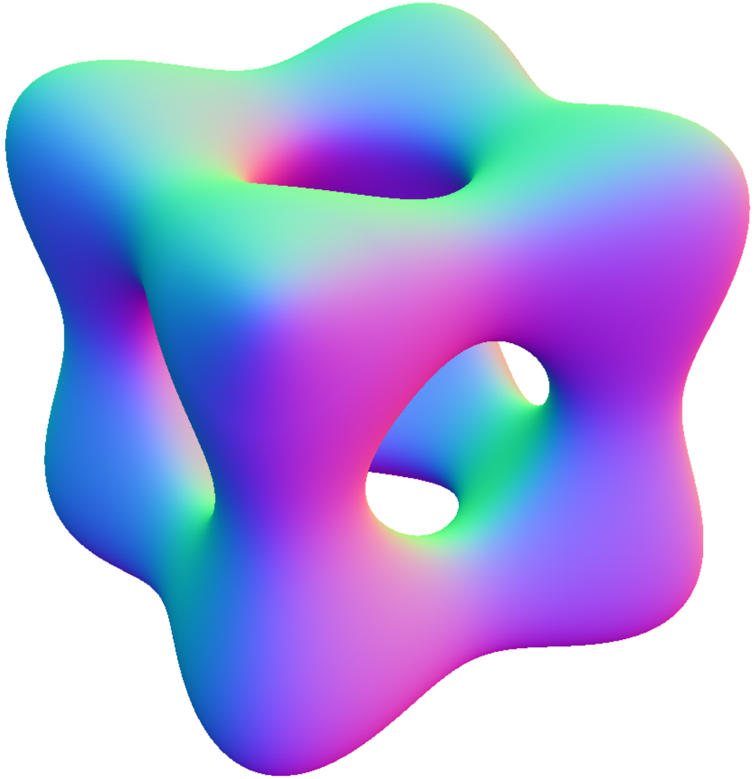
RENDERING IMPLICIT SURFACES IS HARD

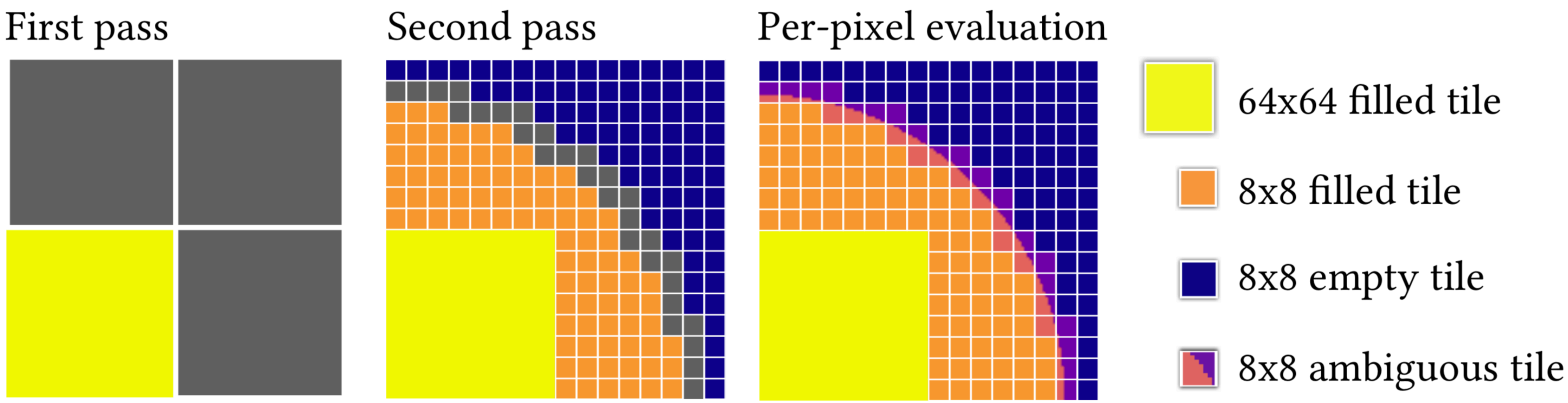
- Meshing
  - Hard to get right for arbitrary shapes
  - Philosophically unsatisfying
- Raytracing
  - Often requires C^1 or Lipshitz continuity
  - Limits the kind of transforms that you can apply



RENDERING IMPLICIT SURFACES IS HARD

- Meshing
  - Hard to get right for arbitrary shapes
  - Philosophically unsatisfying
- Raytracing
  - Often requires C^1 or Lipshitz continuity
  - Limits the kind of transforms that you can apply
- Our strategy
  - Interval arithmetic + subdivision + recursion (Duff ’92)
  - Per-tile command lists on the GPU (Ganacim et al. ’14)
  - Only requires C0 continuity, and scales to large models
  - Extremely philosophically satisfying




First pass
Second pass
Per-pixel evaluation
64x64 filled tile
8x8 filled tile
8x8 empty tile
8x8 ambiguous tile

Interpreter on the GPU

Interval arithmetic

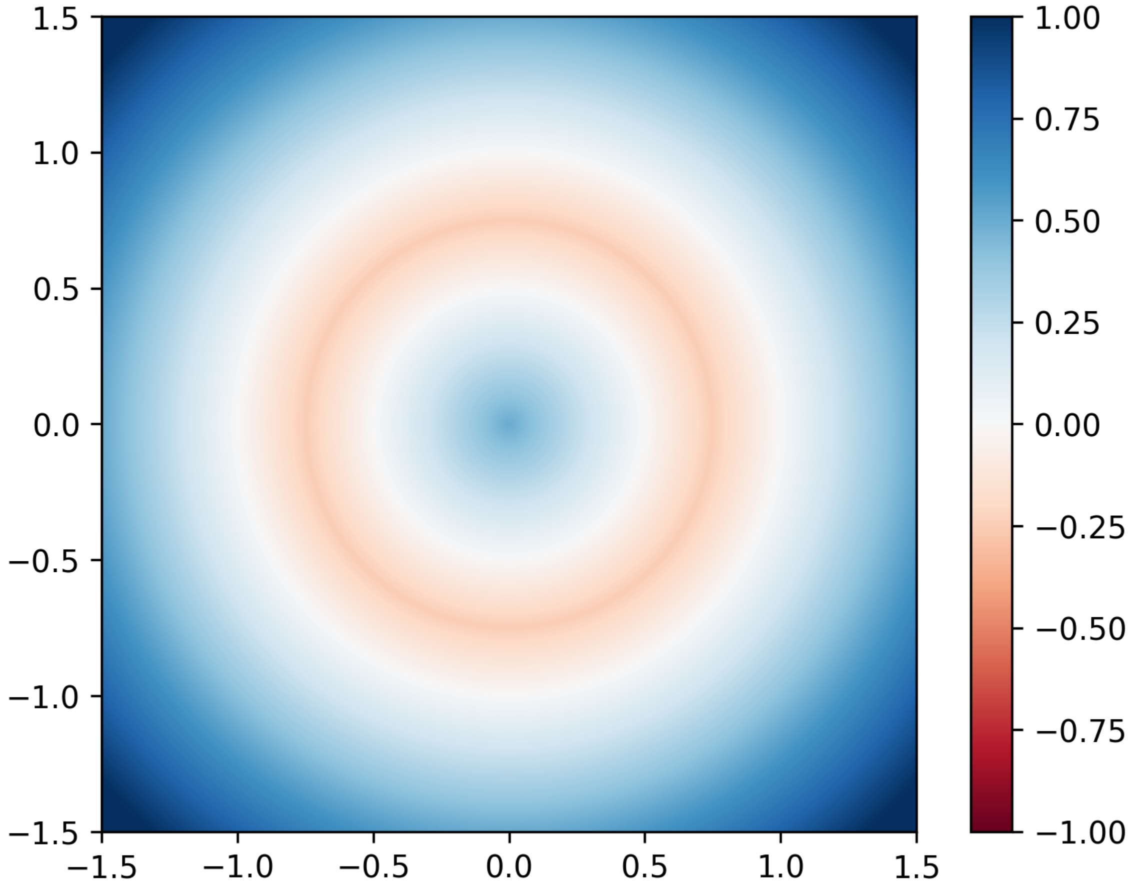
Tape shortening

Interpreter on the GPU

Interval arithmetic

Tape shortening

$$\max\left(0.5 - \sqrt{x^2 + y^2}, \sqrt{x^2 + y^2} - 1\right) < 0$$



Compiled kernel

- Uses hardware registers
- Very fast!
- Longer initial startup (must compile one shader per shape)
- Harder to specialize

Interpreted shape

- Uses soft registers (“slots”)
- Not as fast!
- Faster startup (build tape and send data to GPU)
- Easy to specialize at runtime

Compiled kernel

- Uses hardware registers
- Very fast!
- Longer initial startup (must compile one shader per shape)
- Harder to specialize

Interpreted shape

- Uses soft registers (“slots”)
- Not as fast!
- Faster startup (build tape and send data to GPU)
- **Easy to specialize at runtime**

THE INSTRUCTION TAPE

$$\max\left(0.5 - \sqrt{x^2 + y^2}, \sqrt{x^2 + y^2} - 1\right)$$

opcode	lhs	rhs	out
X	–	–	slot 0
Y	–	–	slot 1
SQUARE	slot 0	–	slot 0
SQUARE	slot 1	–	slot 1
ADD	slot 0	slot 1	slot 1
SQRT	slot 1	–	slot 1
SUB	slot 1	1.0f	slot 0
SUB	0.5f	slot 1	slot 1
MAX	slot 0	slot 1	slot 1

TAPE PACKING

opcode	out	lhs	rhs	immediate
u8	u8	u8	u8	f32

8 bytes per clause

THE INTERPRETER LOOP

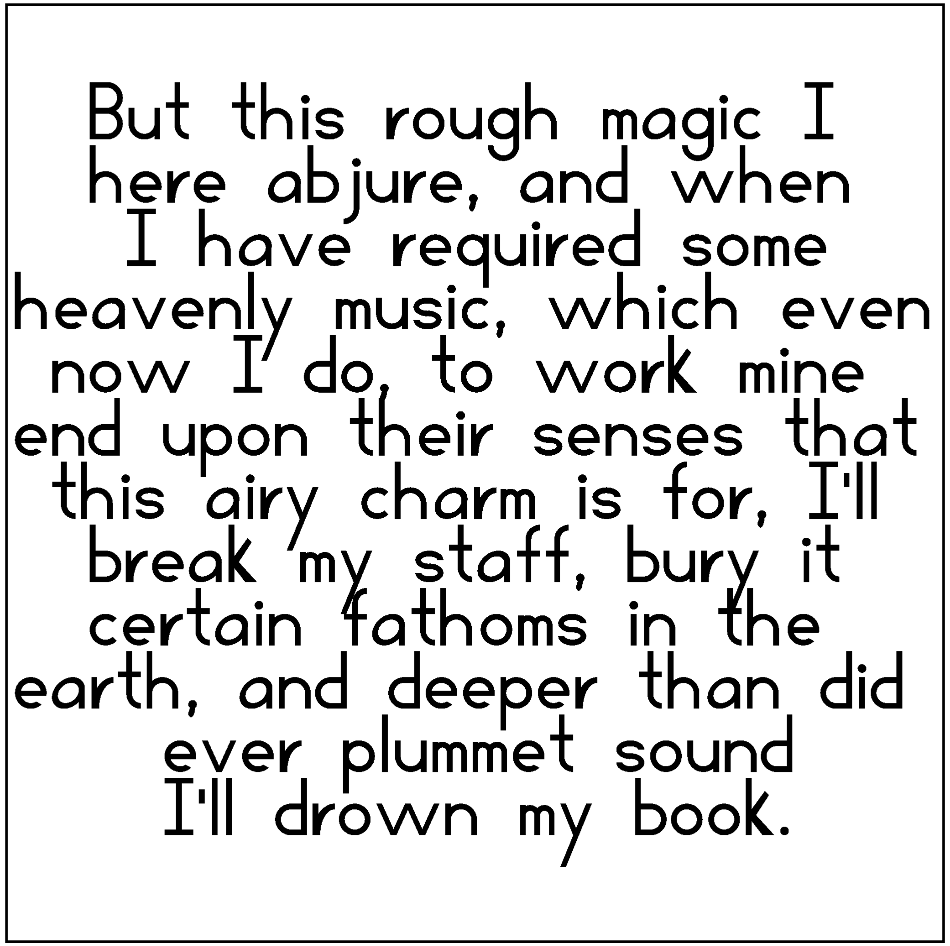
```
def run(tape):
  for (opcode, out, lhs, rhs, imm) in tape:
    match opcode:
      # LHS & RHS
      ADD_LHS_RHS: slots[out] = slots[lhs] + slots[rhs]
      SUB_LHS_RHS: slots[out] = slots[lhs] - slots[rhs]
      MUL_LHS_RHS: slots[out] = slots[lhs] * slots[rhs]
      ...
      # LHS & immediate
      ADD_LHS_IMM: slots[out] = slots[lhs] + imm
      ...and so on...

  return slots[tape[-1].out]
```

INTERPRETER OVERHEAD

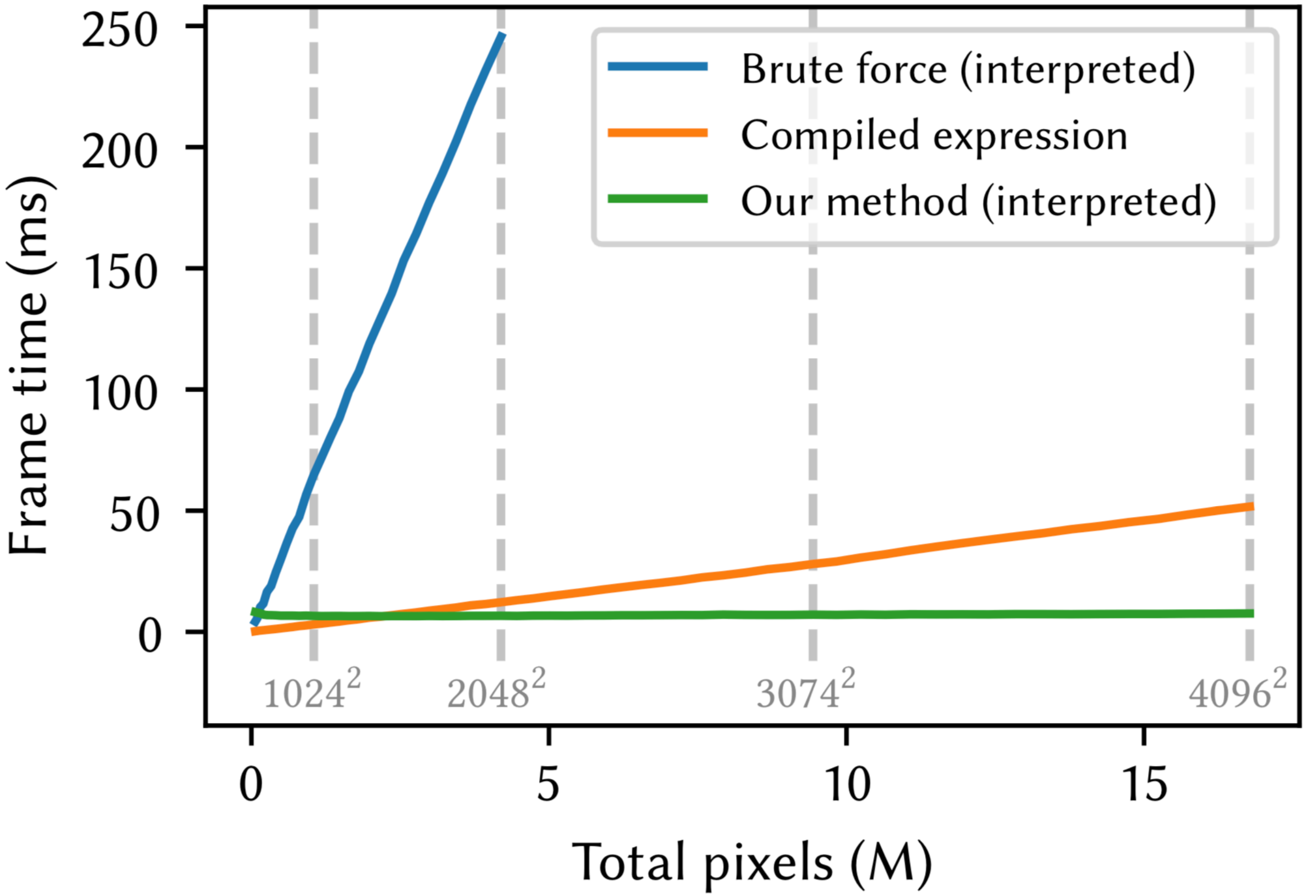
But this rough magic I here abjure, and when I have required some heavenly music, which even now I do, to work mine end upon their senses that this airy charm is for, I'll break my staff, bury it certain fathoms in the earth, and deeper than did ever plummet sound I'll drown my book.

6056 clauses

INTERPRETER OVERHEAD



6056 clauses



GPU PERFORMANCE NOTES

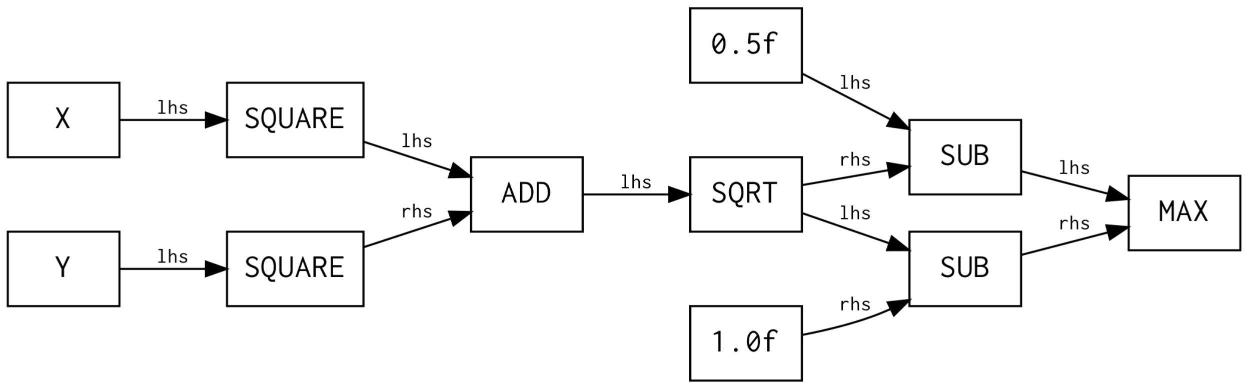
- Implemented in CUDA C/C++, compiled with `nvcc`
- Must use fewer than 32 registers to reach 100% occupancy!
  - Keep inner loop very simple
  - Consider moving setup logic to a separate kernel
  - Prefer C-style over C++-style CUDA
  - Recompile and check after even small changes; the compiler works in mysterious ways
- Avoid thread divergence within a warp
  - Ensure that all threads in a warp are executing the same **tape**
  - 32 threads per warp → 64x subdivision at each level is a convenient multiple

WHERE DOES THE TAPE COME FROM?

$$\max\left(0.5 - \sqrt{x^2 + y^2}, \sqrt{x^2 + y^2} - 1\right)$$

opcode	lhs	rhs	out
X	–	–	slot 0
Y	–	–	slot 1
SQUARE	slot 0	–	slot 0
SQUARE	slot 1	–	slot 1
ADD	slot 0	slot 1	slot 1
SQRT	slot 1	–	slot 1
SUB	slot 1	1.0f	slot 0
SUB	0.5f	slot 1	slot 1
MAX	slot 0	slot 1	slot 1

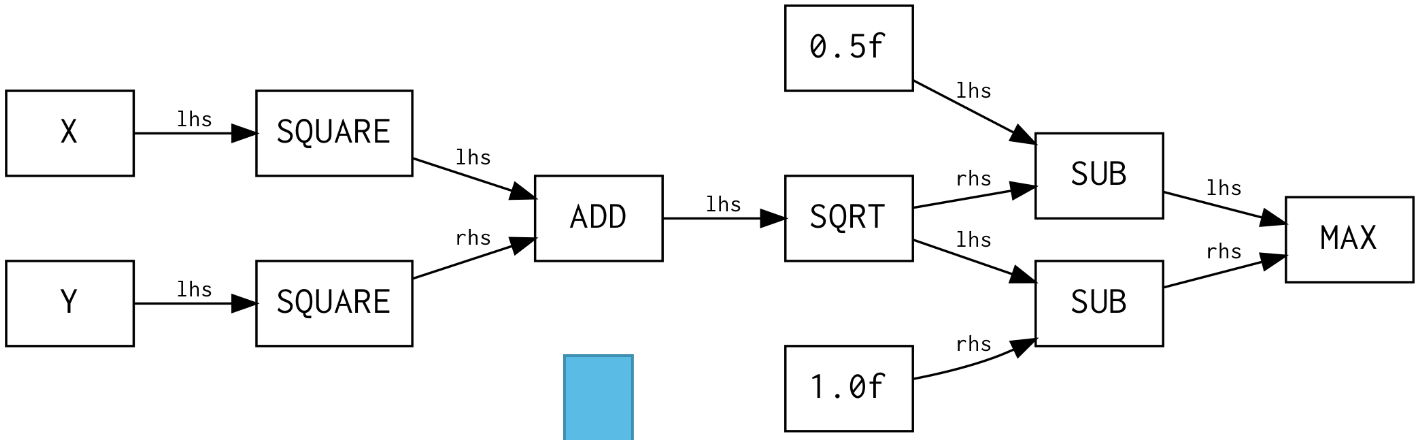
DAG → INSTRUCTION TAPE

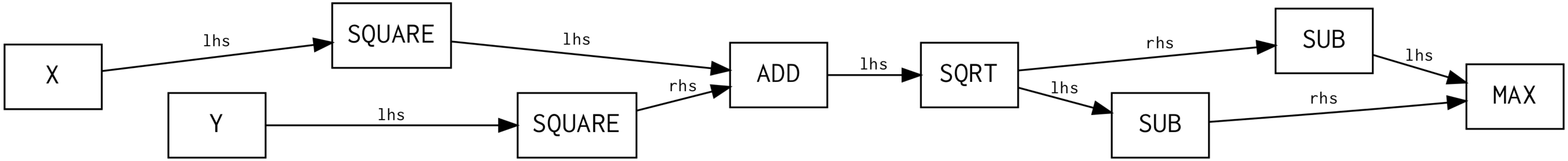


- The equation is encoded as a directed acyclic graph
- Common subexpressions are merged into one node

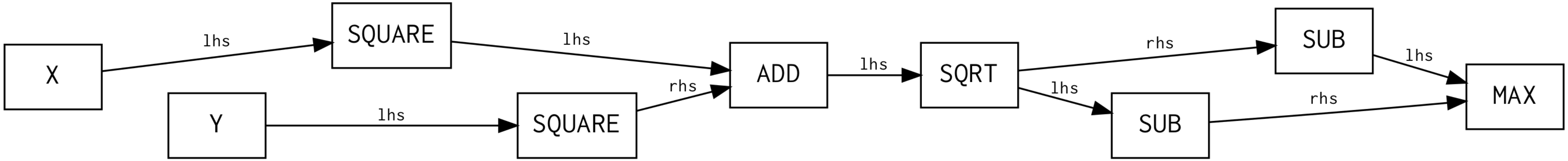
opcode	lhs	rhs	out
X	–	–	slot 0
Y	–	–	slot 1
SQUARE	slot 0	–	slot 0
SQUARE	slot 1	–	slot 1
ADD	slot 0	slot 1	slot 1
SQRT	slot 1	–	slot 1
SUB	slot 1	1.0f	slot 0
SUB	0.5f	slot 1	slot 1
MAX	slot 0	slot 1	slot 1

TOPOLOGICAL SORT





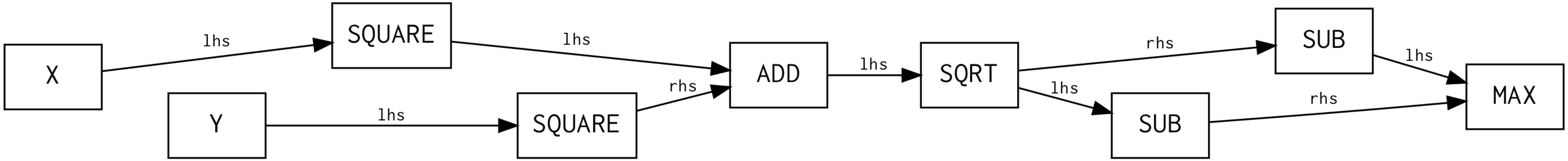
SLOT ALLOCATION



- Same problem as **register allocation** in compiler design
- Simple greedy algorithm:
  – Maintain a pool of available slots
  – After a slot is used for the last time, release it to the pool
  – Choose the top slot in the pool
  – If pool is empty, create a new slot

opcode	lhs	rhs	out
X	–	–	\$1
Y	–	–	\$2
SQUARE	\$1	–	\$3
SQUARE	\$2	–	\$4
ADD	\$3	\$4	\$5
SQRT	\$5	–	\$6
SUB	\$6	1.0f	\$7
SUB	0.5f	\$6	\$8
MAX	\$7	\$8	\$9

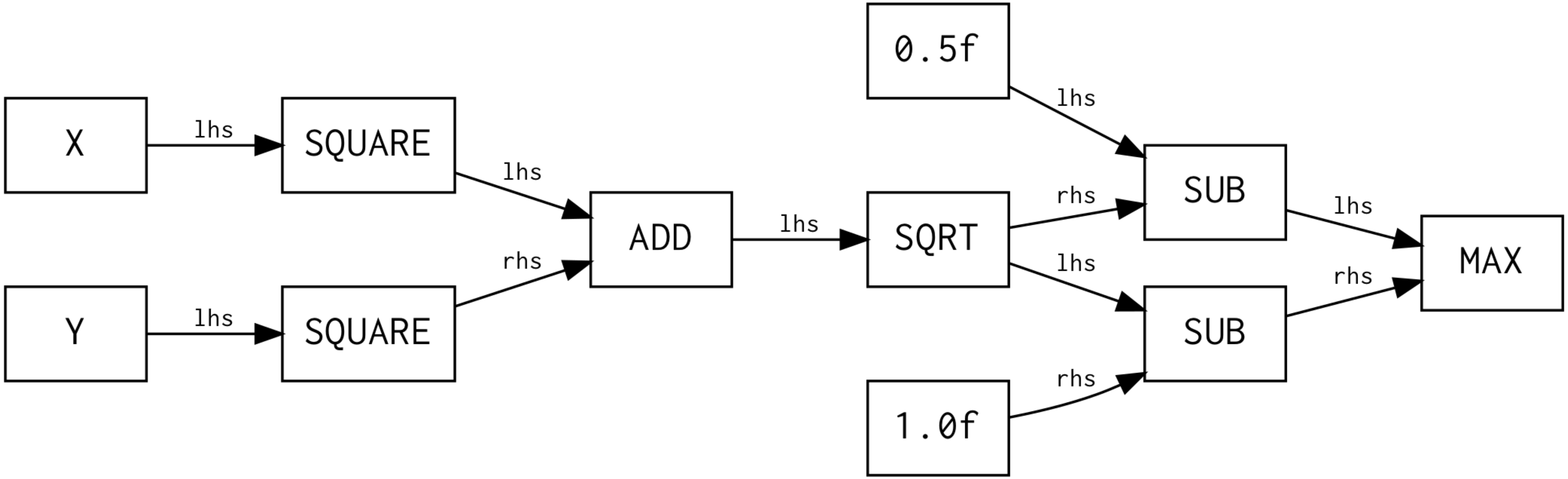
SLOT ALLOCATION



- Same problem as **register allocation** in compiler design
- Simple greedy algorithm:
  - Maintain a pool of available slots
  - After a slot is used for the last time, release it to the pool
  - Choose the top slot in the pool
  - If pool is empty, create a new slot

opcode	lhs	rhs	out
X	–	–	slot 0
Y	–	–	slot 1
SQUARE	slot 0	–	slot 0
SQUARE	slot 1	–	slot 1
ADD	slot 0	slot 1	slot 1
SQRT	slot 1	–	slot 1
SUB	slot 1	1.0f	slot 0
SUB	0.5f	slot 1	slot 1
MAX	slot 0	slot 1	slot 1

WHERE DOES THE DAG COME FROM?



Studio
Desktop design tool

Bindings
Scheme and Python

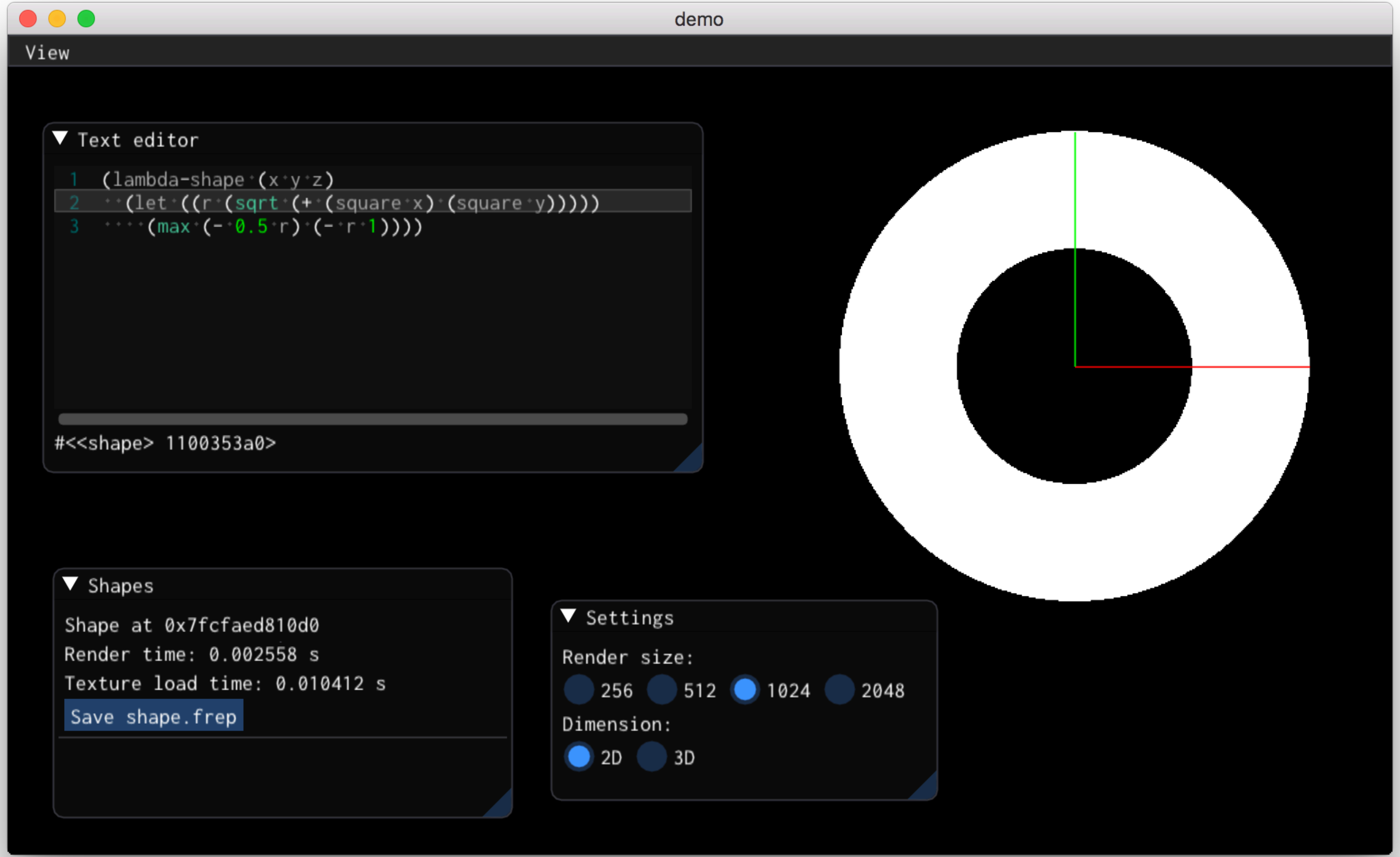
Standard library
Shapes and geometric operations

C API
#include <libfive.h>

libfive
C++ library

libfive

https://libfive.com


demo
View
Text editor
(lambda-shape (x y z)
(let ((r (sqrt (+ (square x) (square y)))))
(max (- 0.5 r) (- r 1))))
#<<shape> 1100353a0>
Shapes
Shape at 0x7fcfaed810d0
Render time: 0.002558 s
Texture load time: 0.010412 s
Save shape.frep
Settings
Render size:
256
512
1024
2048
Dimension:
2D
3D

Interpreter on the GPU

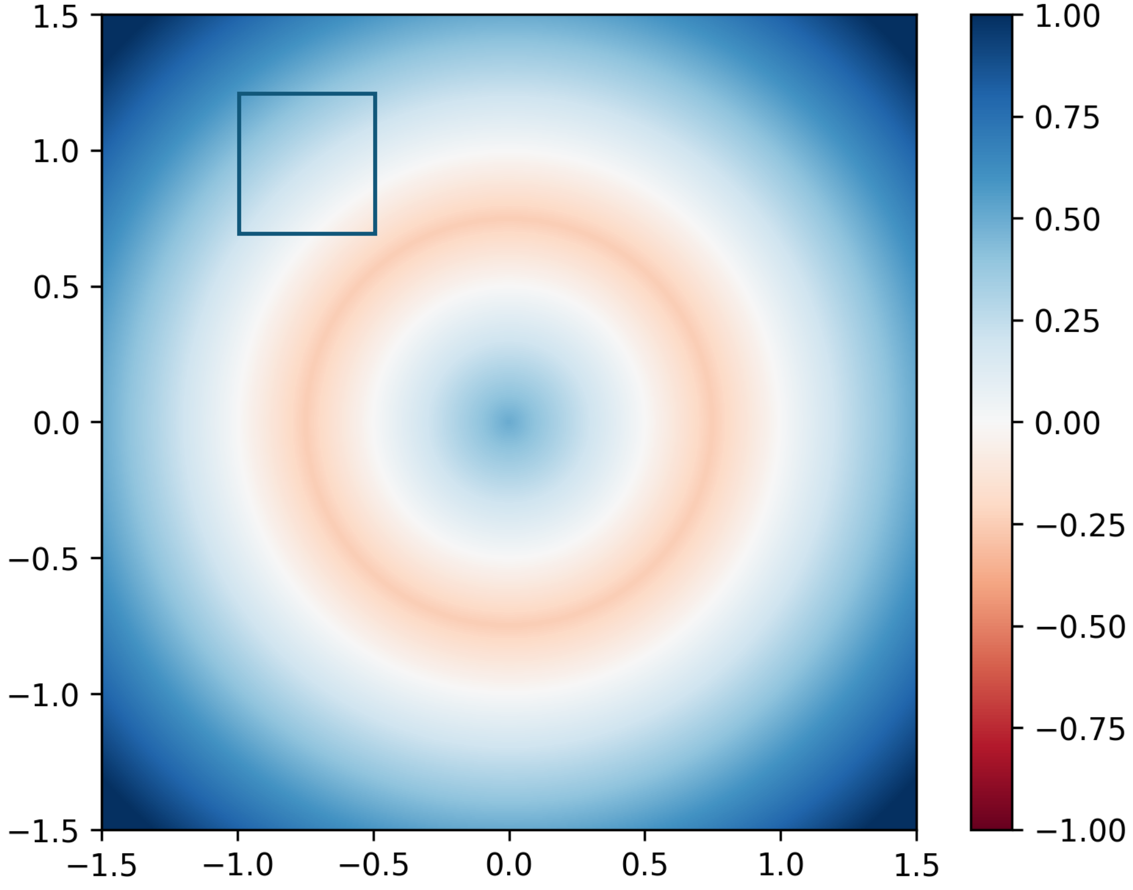
Interval arithmetic

Tape shortening

Interpreter on the GPU

Interval arithmetic

Tape shortening


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

INTERVAL ARITHMETIC

opcode	lhs	rhs	out
X	–	–	[-1.0, -0.5]
Y	–	–	[0.7, 1.2]
SQUARE		–	
SQUARE		–	
ADD			
SQRT		–	
SUB		1.0f	
SUB	0.5f		
MAX			

INTERVAL ARITHMETIC

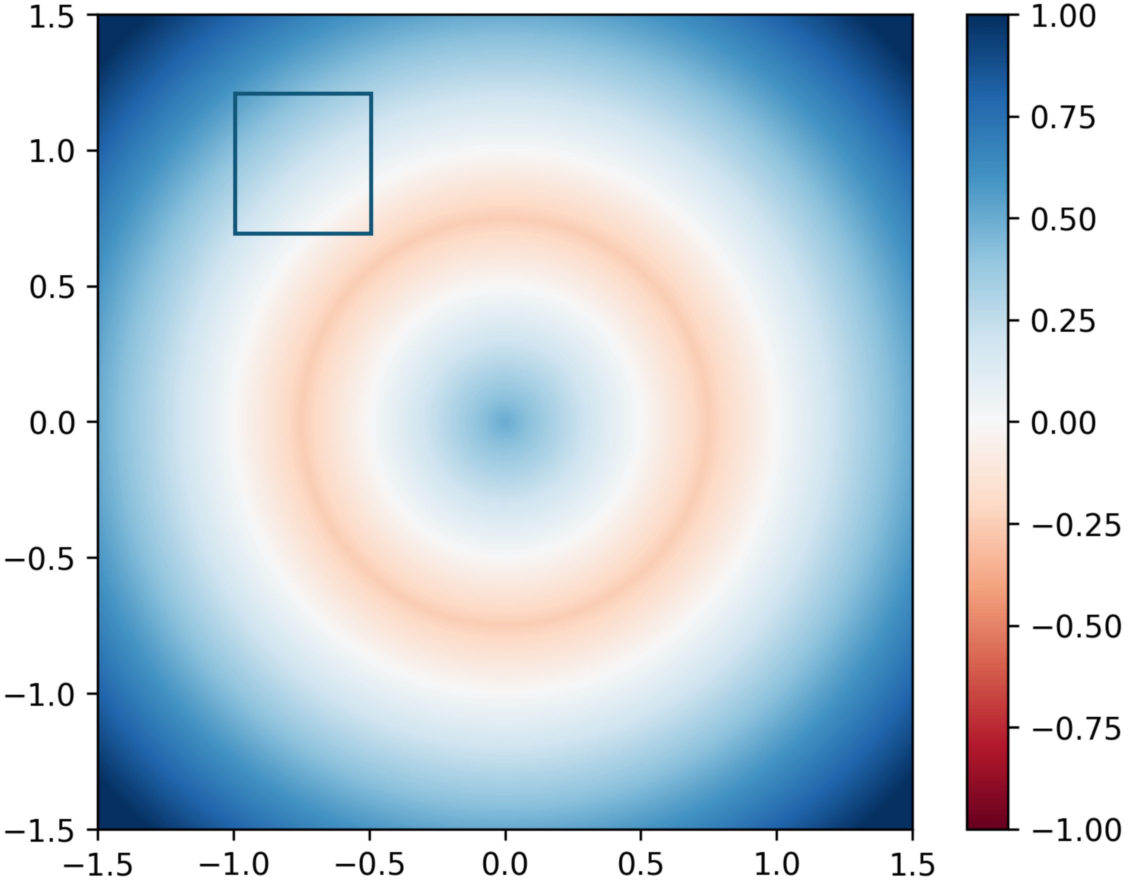
opcode	lhs	rhs	out
X	–	–	[-1.0, -0.5]
Y	–	–	[0.7, 1.2]
SQUARE	[-1.0, -0.5]	–	
SQUARE		–	
ADD			
SQRT		–	
SUB		1.0f	
SUB	0.5f		
MAX			

INTERVAL ARITHMETIC

opcode	lhs	rhs	out
X	–	–	[-1.0, -0.5]
Y	–	–	[0.7, 1.2]
SQUARE	[-1.0, -0.5]	–	[0.25, 1.0]
SQUARE		–	
ADD			
SQRT		–	
SUB		1.0f	
SUB	0.5f		
MAX			

INTERVAL ARITHMETIC

opcode	lhs	rhs	out
X	–	–	[-1.0, -0.5]
Y	–	–	[0.7, 1.2]
SQUARE	[-1.0, -0.5]	–	[0.25, 1.0]
SQUARE	[0.7, 1.2]	–	[0.49, 1.44]
ADD	[0.25, 1.0]	[0.49, 1.44]	[0.74, 2.44]
SQRT	[0.74, 2.44]	–	[0.86, 1.56]
SUB	[0.86, 1.56]	1.0f	[-0.14, 0.56]
SUB	0.5f	[0.86, 1.56]	[-1.06, -0.36]
MAX	[-0.14, 0.56]	[-1.06, -0.36]	**[-0.14, 0.56]**



1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

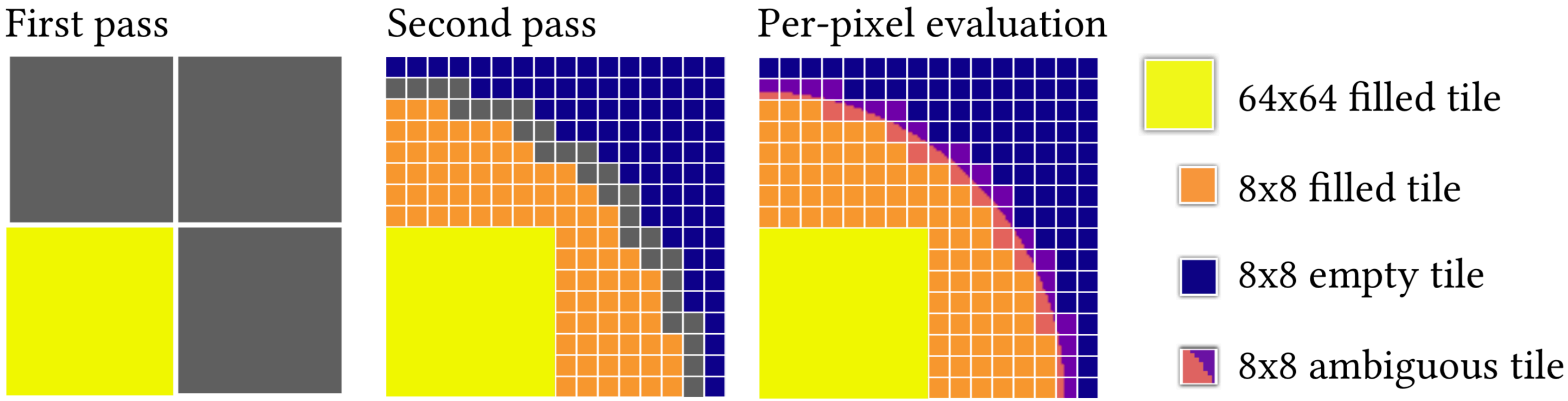
INTERVAL ARITHMETIC RESULTS

```
f(X, Y, Z).upper < 0
f(X, Y, Z).lower > 0
           otherwise
```

`f(X, Y, Z).upper < 0` → inside the shape (“filled”)

`f(X, Y, Z).lower > 0` → outside the shape (“empty”)

`otherwise` → ambiguous


First pass
Second pass
Per-pixel evaluation
64x64 filled tile
8x8 filled tile
8x8 empty tile
8x8 ambiguous tile

GPU IMPLEMENTATION NOTES

- Implemented a GPU interval class based on `boost::interval`
- Using compiler intrinsics to ensure proper rounding modes

```
__device__ inline Interval operator+(const Interval& x,
                                     const Interval& y)
{
    return {__fadd_rd(x.lower(), y.lower()),
            __fadd_ru(x.upper(), y.upper())};
}
```

- Intrinsics are not available on all GPU compute platforms (but it may not matter)

Interpreter on the GPU

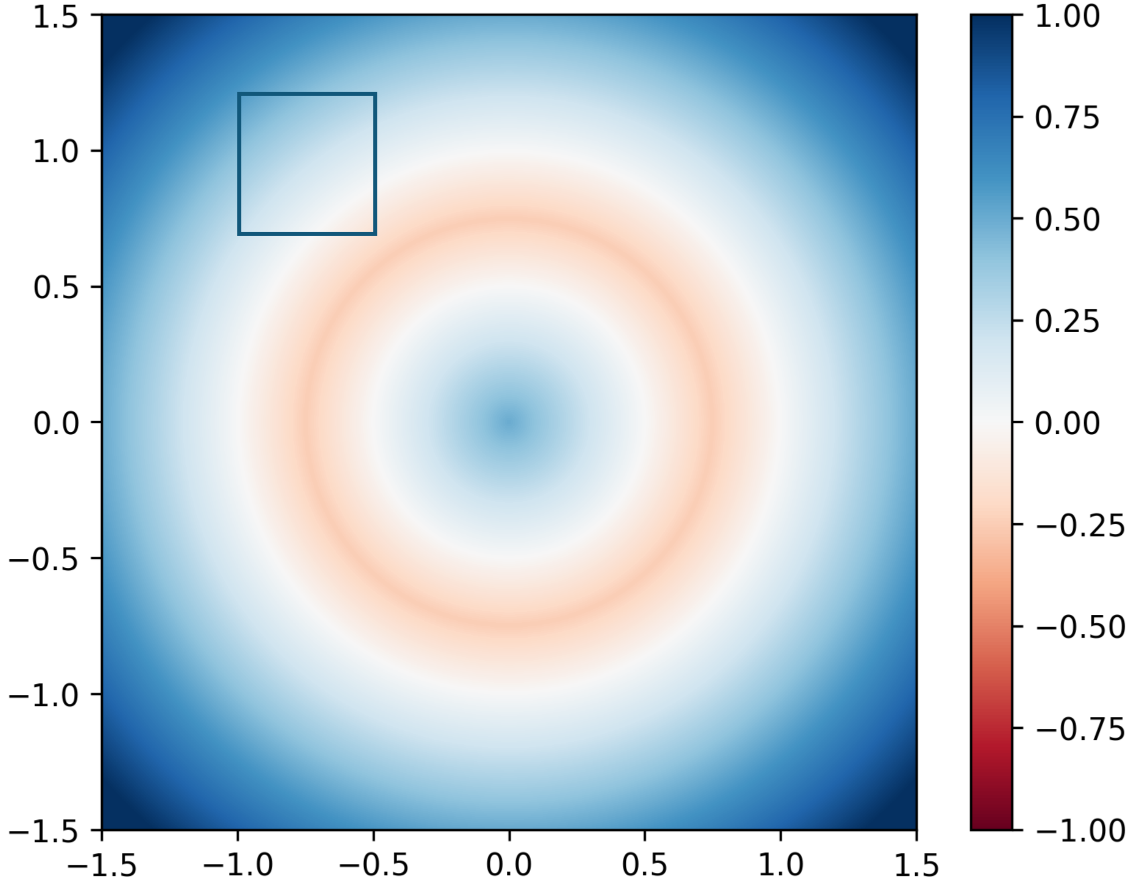
Interval arithmetic

Tape shortening

Interpreter on the GPU

Interval arithmetic

Tape shortening

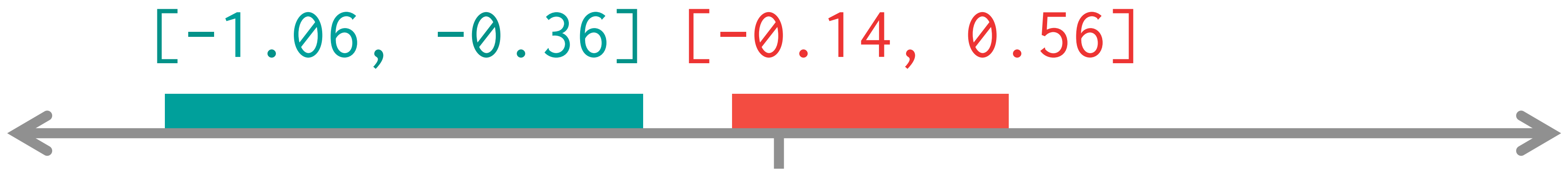


$$\max\left(0.5 - \sqrt{x^2 + y^2}, \sqrt{x^2 + y^2} - 1\right)$$

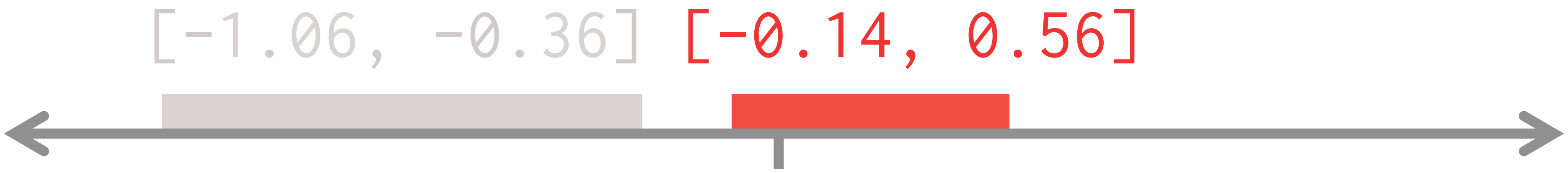
MIN AND MAX ARE SPECIAL

opcode	lhs	rhs	out
X	–	–	[-1.0, -0.5]
Y	–	–	[0.7, 1.2]
SQUARE	[-1.0, -0.5]	–	[0.25, 1.0]
SQUARE	[0.7, 1.2]	–	[0.49, 1.44]
ADD	[0.25, 1.0]	[0.49, 1.44]	[0.74, 2.44]
SQRT	[0.74, 2.44]	–	[0.86, 1.56]
SUB	[0.86, 1.56]	1.0f	[-0.14, 0.56]
SUB	0.5f	[0.86, 1.56]	[-1.06, -0.36]
MAX	[-0.14, 0.56]	[-1.06, -0.36]	[-0.14, 0.56]

MIN AND MAX ARE SPECIAL



MIN AND MAX ARE SPECIAL



REPLACING WITH COPY

opcode	lhs	rhs	out
X	–	–	slot 0
Y	–	–	slot 1
SQUARE	slot 0	–	slot 0
SQUARE	slot 1	–	slot 1
ADD	slot 0	slot 1	slot 1
SQRT	slot 1	–	slot 1
SUB	slot 1	1.0f	slot 0
SUB	0.5f	slot 1	slot 1
MAX	slot 0	slot 1	slot 1

REPLACING WITH COPY

opcode	lhs	rhs	out
X	–	–	slot 0
Y	–	–	slot 1
SQUARE	slot 0	–	slot 0
SQUARE	slot 1	–	slot 1
ADD	slot 0	slot 1	slot 1
SQRT	slot 1	–	slot 1
SUB	slot 1	1.0f	slot 0
SUB	0.5f	slot 1	slot 1
COPY	slot 0	–	slot 1

NOTICING INACTIVE CLAUSES

opcode	lhs	rhs	out
X	–	–	slot 0
Y	–	–	slot 1
SQUARE	slot 0	–	slot 0
SQUARE	slot 1	–	slot 1
ADD	slot 0	slot 1	slot 1
SQRT	slot 1	–	slot 1
SUB	slot 1	1.0f	slot 0
~~SUB~~	~~0.5f~~	~~slot 1~~	~~slot 1~~
COPY	slot 0	–	slot 1

NOTICING INACTIVE CLAUSES

opcode	lhs	rhs	out
X	–	–	slot 0
Y	–	–	slot 1
SQUARE	slot 0	–	slot 0
SQUARE	slot 1	–	slot 1
ADD	slot 0	slot 1	slot 1
SQRT	slot 1	–	slot 1
SUB	slot 1	1.0f	slot 0
COPY	slot 0	–	slot 1

TAPE SHORTENING

- Use interval results at `min` / `max` clauses to detect clauses that are inactive in a particular region
- Construct a shortened tape without those clauses
- That tape is valid for all subregions contained within the parent region

Algorithm 1: Evaluate a tape, recording which branch is taken at every `min` and `max` node

```
choices ← an empty stack
foreach clause in tape do
    lhs ← getValue(clause.lhs)
    rhs ← getValue(clause.rhs)
    switch clause.opcode do
        case OP_MIN
            if lhs.upper < rhs.lower then
                choices.push(CHOICE_LHS)
            else if rhs.upper < lhs.lower then
                choices.push(CHOICE_RHS)
            else
                choices.push(CHOICE_BOTH)
            slots[clause.out] ← min(lhs, rhs)
        case OP_MAX
            Similar logic to push a choice slots[clause.out]
            ← max(lhs, rhs)
        case OP_ADD
            slots[clause.out] ← lhs + rhs
        case OP_SUB
            ...and so on for other opcodes
clause ← the last clause in the tape
return (slots[clause.out], choices)
```

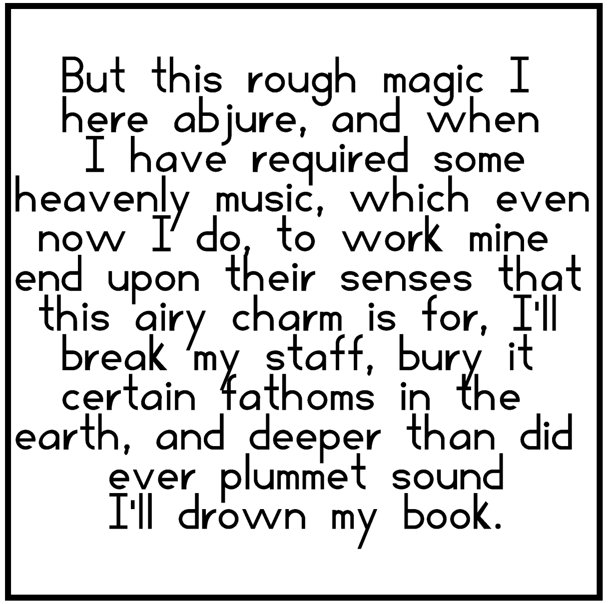
Algorithm 2: Use the choice data from Alg. 1 to construct a shorter tape which only contains active clauses

```
output ← an empty tape
active ← an array of all false
active[final output slot] ← true
foreach clause in tape.reversed() do
    if clause.opcode ∈ [OP_MIN, OP_MAX] then
        choice ← choices.pop()
    else
        choice ← CHOICE_BOTH
    if active[clause.out] then
        active[clause.out] ← false
        if choice == CHOICE_LHS then
            active[clause.lhs] ← true
            clause.rhs ← clause.lhs
        else if choice == CHOICE_RHS then
            active[clause.rhs] ← true
            clause.lhs ← clause.rhs
        else
            active[clause.lhs] ← true
            active[clause.rhs] ← true
        output.push_back(clause)
return output
```

TAPE SHORTENING

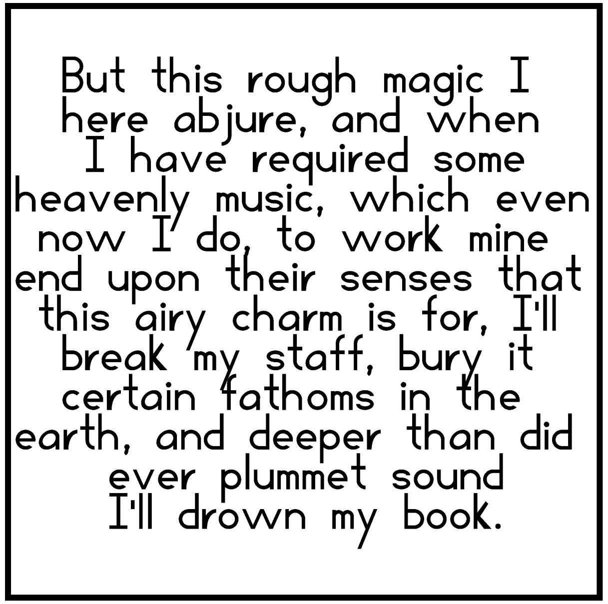
- Analogous to *mark-and-sweep* garbage collection
- Mark output slot as **active**, all other slots as **inactive**
- Walk *backwards* through the tape, skipping clauses with **inactive** output slots
- For `MIN` and `MAX` clauses:
  - Mark the output slot as **inactive**
  - If unambiguous, only mark *one* input output slot as **active**, and replace `MIN/MAX` with `COPY`
  - Otherwise, mark *all* input slots as **active**
- For all **other** clauses
  - Mark the output slot as **inactive**
  - Mark *all* input slots as **active**

HOW WELL DOES THIS WORK?

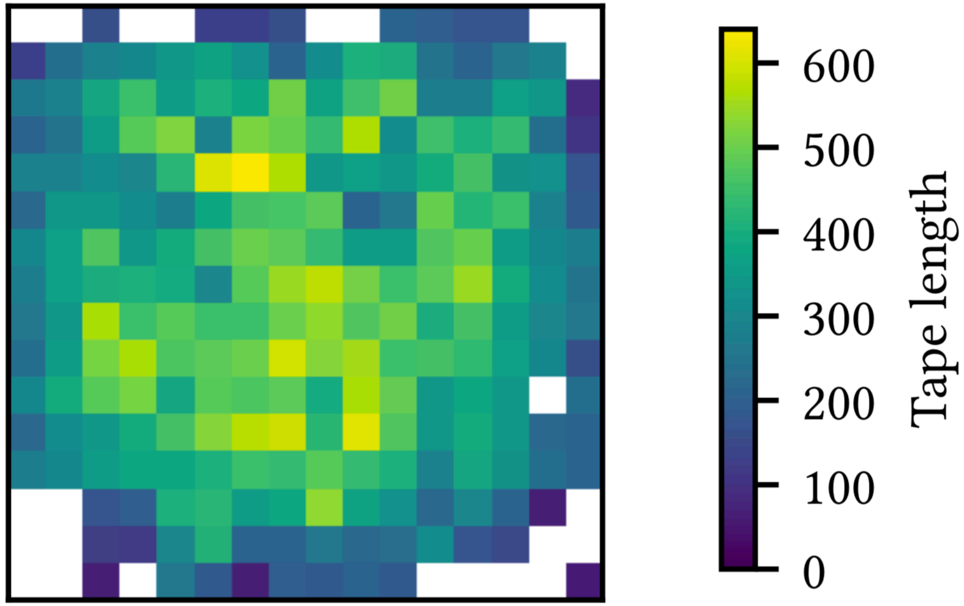


Original: 6056 clauses
(rendered at 1024x1024)

HOW WELL DOES THIS WORK?

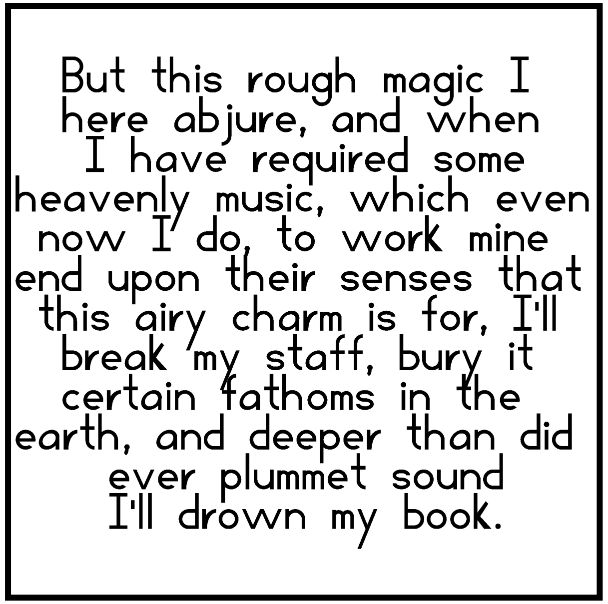


Original: 6056 clauses
(rendered at 1024x1024)

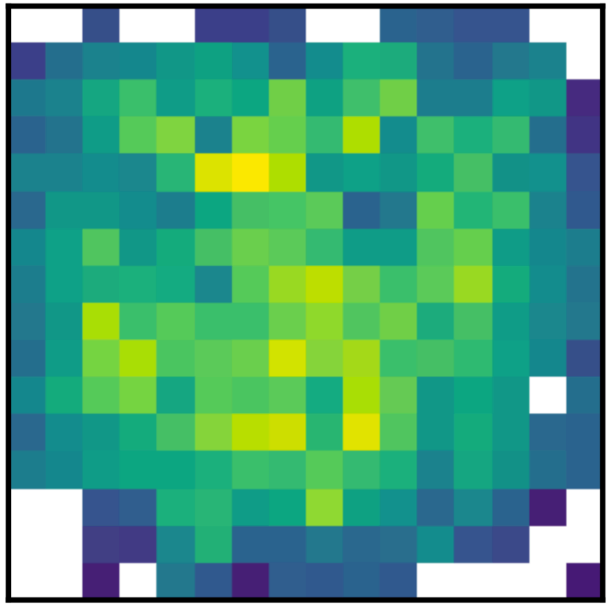


356 ± 125 clauses

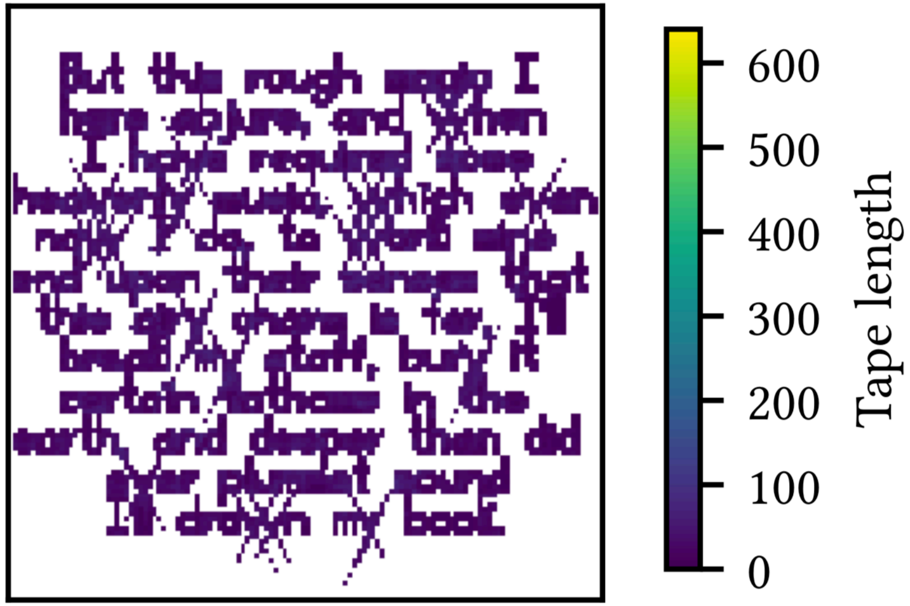
HOW WELL DOES THIS WORK?



Original: 6056 clauses
(rendered at 1024x1024)



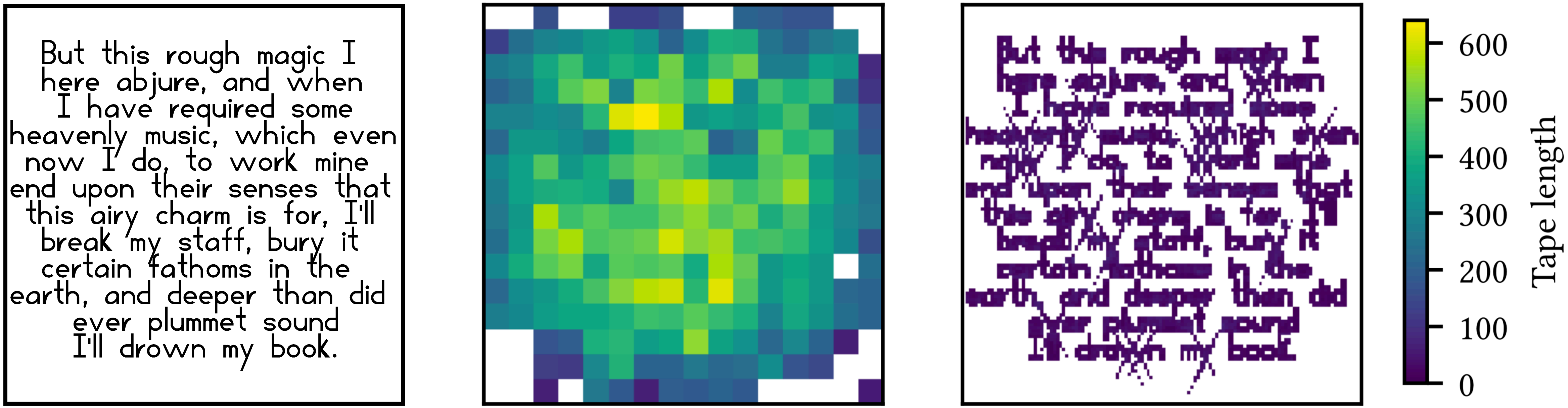
356 ± 125 clauses



28 ± 13 clauses

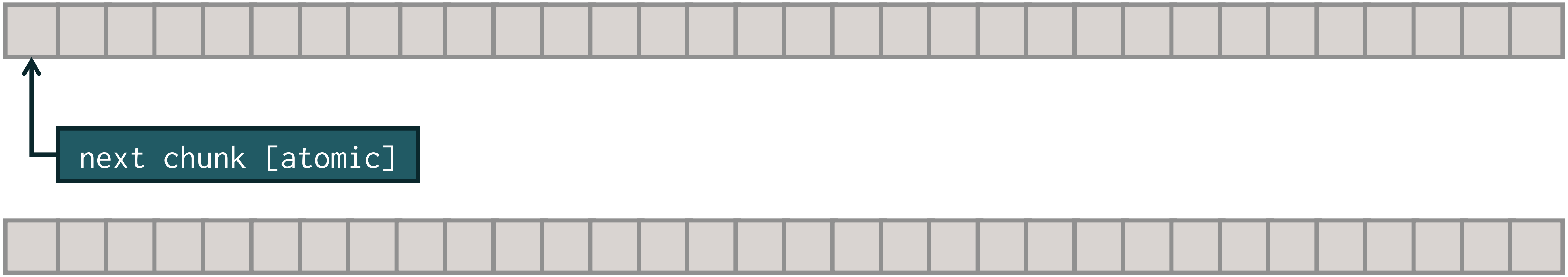
SHORTENING TAPES IN PRACTICE

- We no longer have a single input tape!
- Each region can build a shortened tape
- Dynamic memory allocation on the GPU?



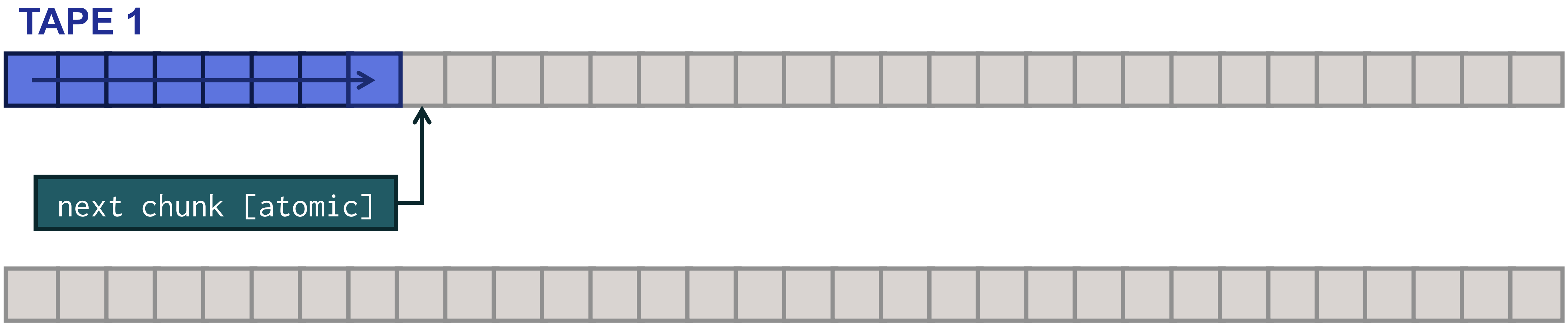
STORING TAPES ON THE GPU

- Unrolled linked lists (64 clauses per chunk)
- Stored in a large (> 1 GB) pre-allocated array in **global** memory
- Threads claim memory by bumping an atomic integer



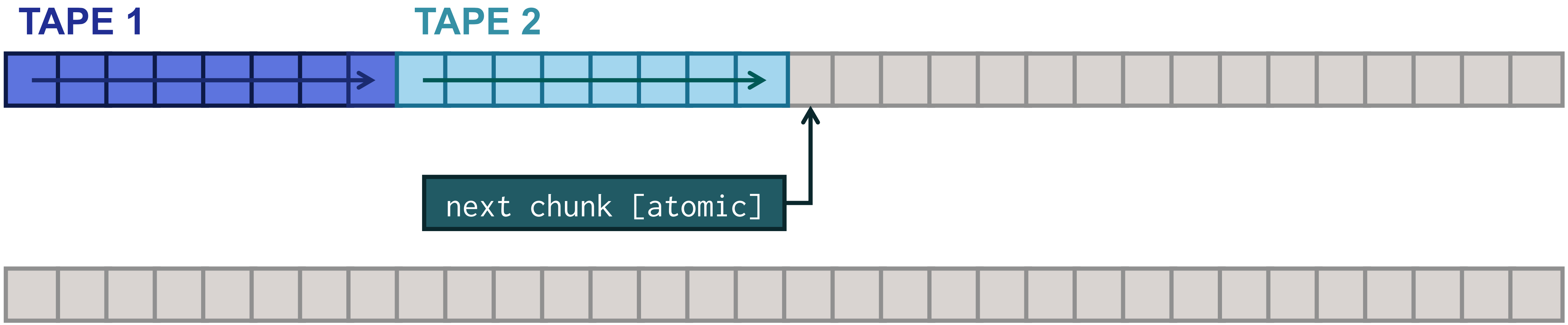
STORING TAPES ON THE GPU

- Unrolled linked lists (64 clauses per chunk)
- Stored in a large (> 1 GB) pre-allocated array in **global** memory
- Threads claim memory by bumping an atomic integer



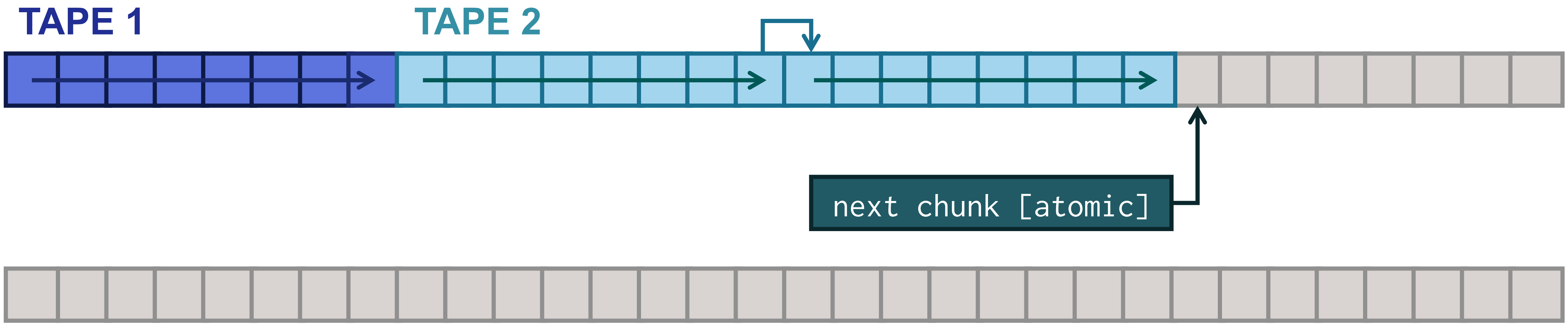
STORING TAPES ON THE GPU

- Unrolled linked lists (64 clauses per chunk)
- Stored in a large (> 1 GB) pre-allocated array in **global** memory
- Threads claim memory by bumping an atomic integer



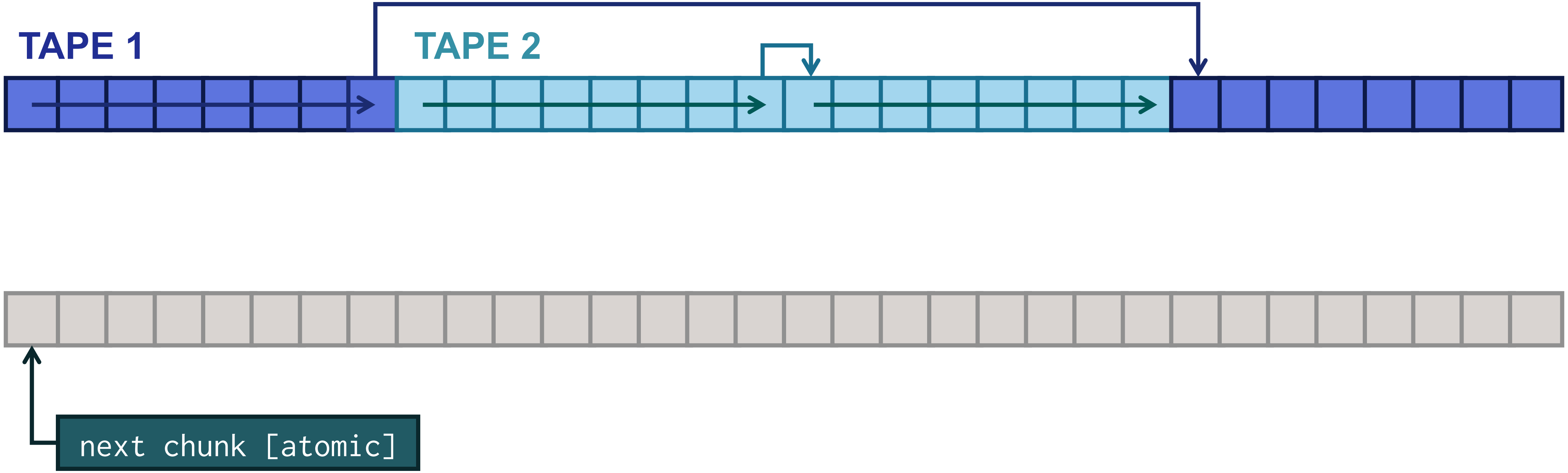
STORING TAPES ON THE GPU

- Unrolled linked lists (64 clauses per chunk)
- Stored in a large (> 1 GB) pre-allocated array in **global** memory
- Threads claim memory by bumping an atomic integer



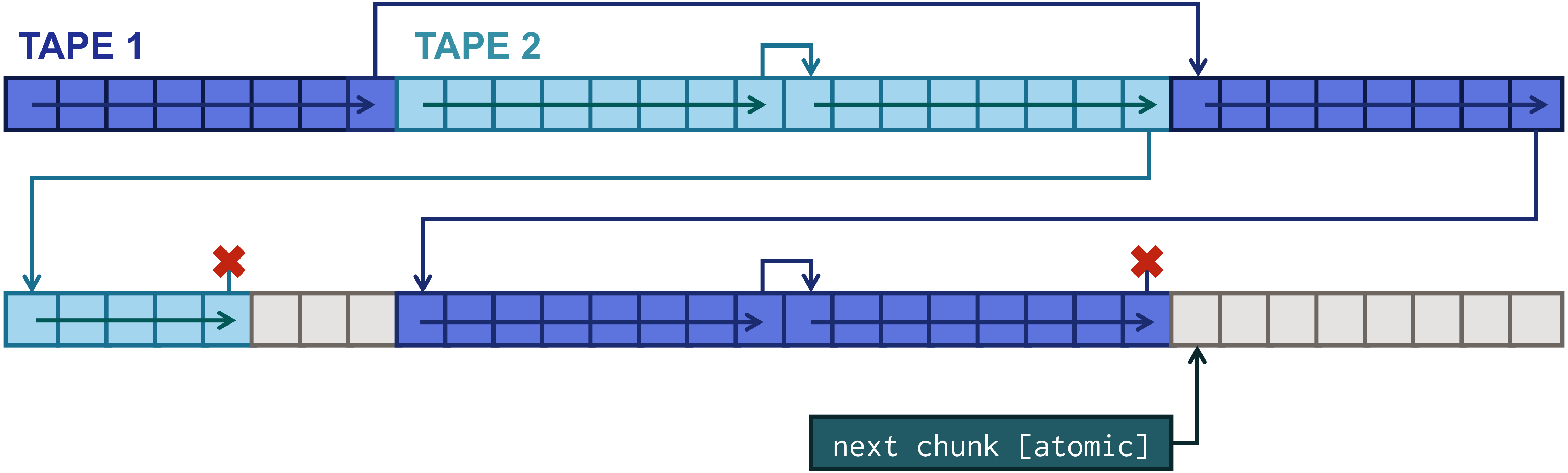
STORING TAPES ON THE GPU

- Unrolled linked lists (64 clauses per chunk)
- Stored in a large (> 1 GB) pre-allocated array in **global** memory
- Threads claim memory by bumping an atomic integer



STORING TAPES ON THE GPU

- Unrolled linked lists (64 clauses per chunk)
- Stored in a large (> 1 GB) pre-allocated array in **global** memory
- Threads claim memory by bumping an atomic integer



opcode	out	lhs	rhs	immediate *or jump*
u8	u8	u8	u8	f32 / i32
JUMP	—	—	—	128

THE INTERPRETER LOOP

```
def run(tape):
  i = 0
  loop:
    (opcode, lhs, rhs, out, imm) = tape[i]
    match opcode:
      DONE: break
      JUMP: i += imm
      ADD_LHS_RHS: slots[out] = slots[lhs] + slots[rhs]
      ...and so on...
    i += 1
  ...
```

Interpreter on the GPU

Interval arithmetic

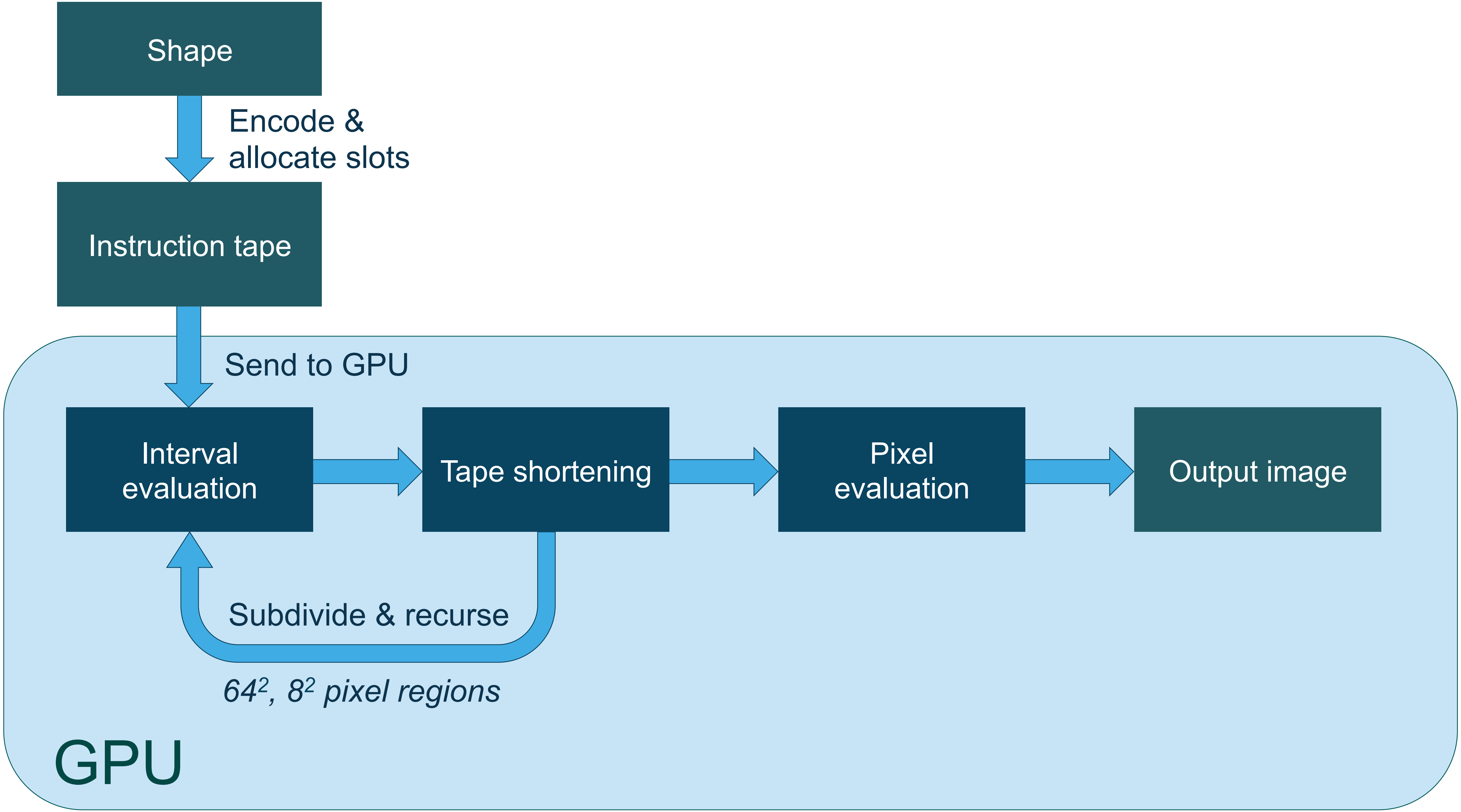
Tape shortening

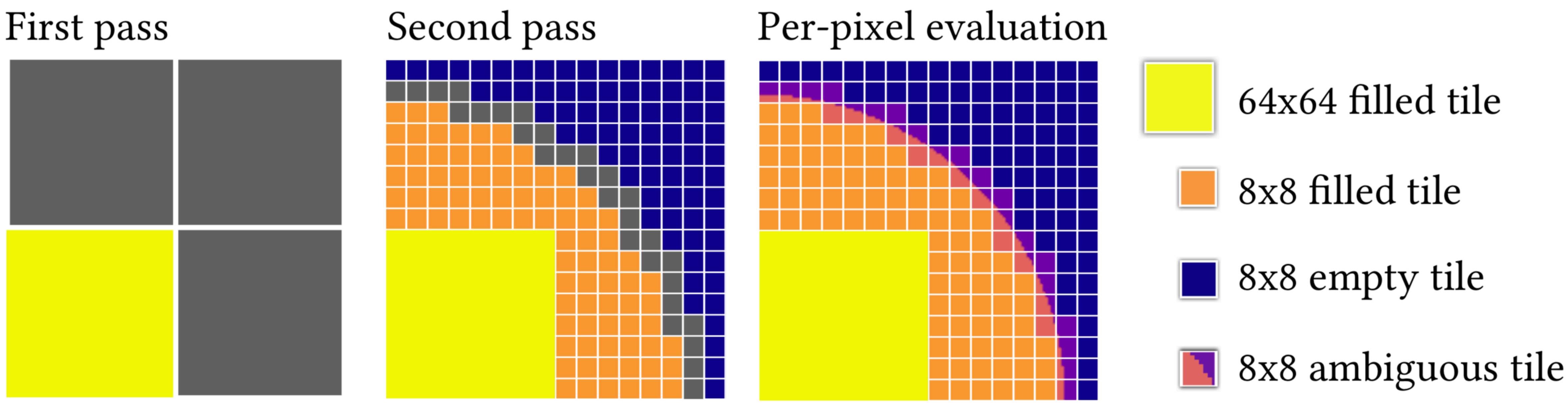
Interpreter on the GPU

Interval arithmetic

Tape shortening

PUTTING IT ALL TOGETHER!

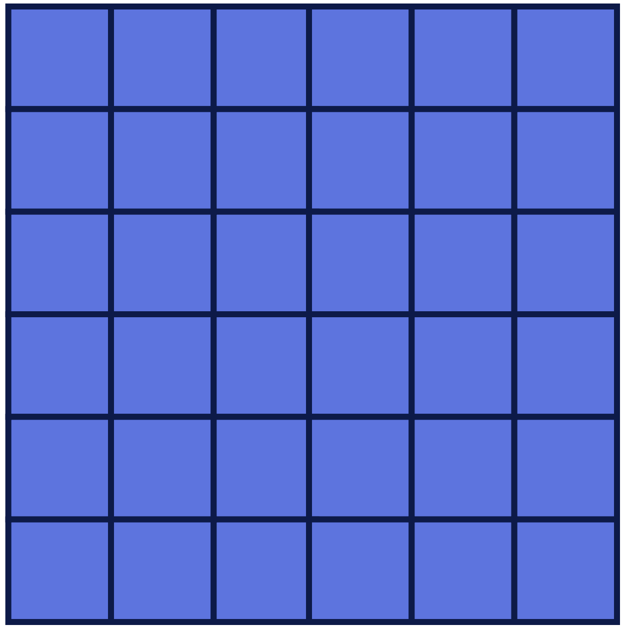

Shape
Encode &
allocate slots
Instruction tape
Send to GPU
Interval evaluation
Tape shortening
Pixel evaluation
Output image
Subdivide & recurse
64², 8² pixel regions
GPU


First pass
Second pass
Per-pixel evaluation
64x64 filled tile
8x8 filled tile
8x8 empty tile
8x8 ambiguous tile

2D RENDERING PROCEDURE

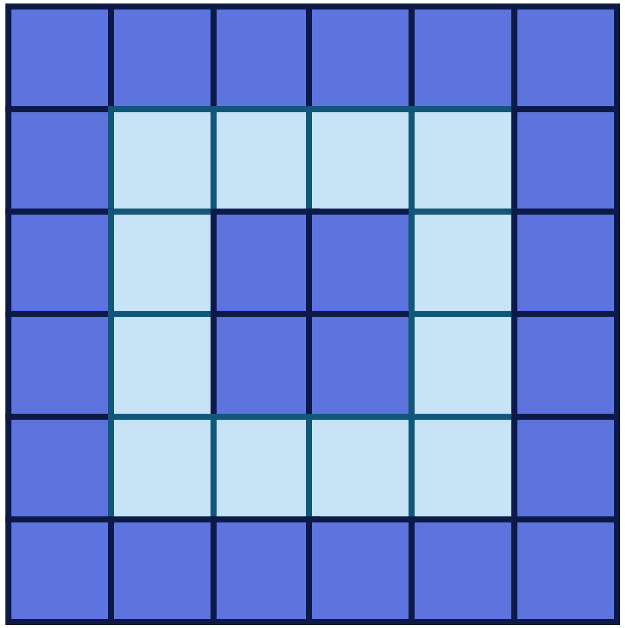
- Interval evaluation of 64x64-pixel **tiles**
  - Record empty and filled tiles, but do nothing more with them
  - For ambiguous tiles, calculate shortened tape
  - Collect ambiguous tiles into a dense list for further evaluation
- Split ambiguous tiles into 64 **subtiles** (8x8-pixels) each
- Interval evaluation of subtiles
  - Record empty and filled subtiles, but do nothing more with them
  - For ambiguous subtiles, calculate shortened tape
  - Collect ambiguous subtiles into a dense list for pixel evaluation
- Per-**pixel** evaluations of ambiguous subtiles
  - Same interpreter loop as interval evaluation, but using single values instead

MEMORY LAYOUT



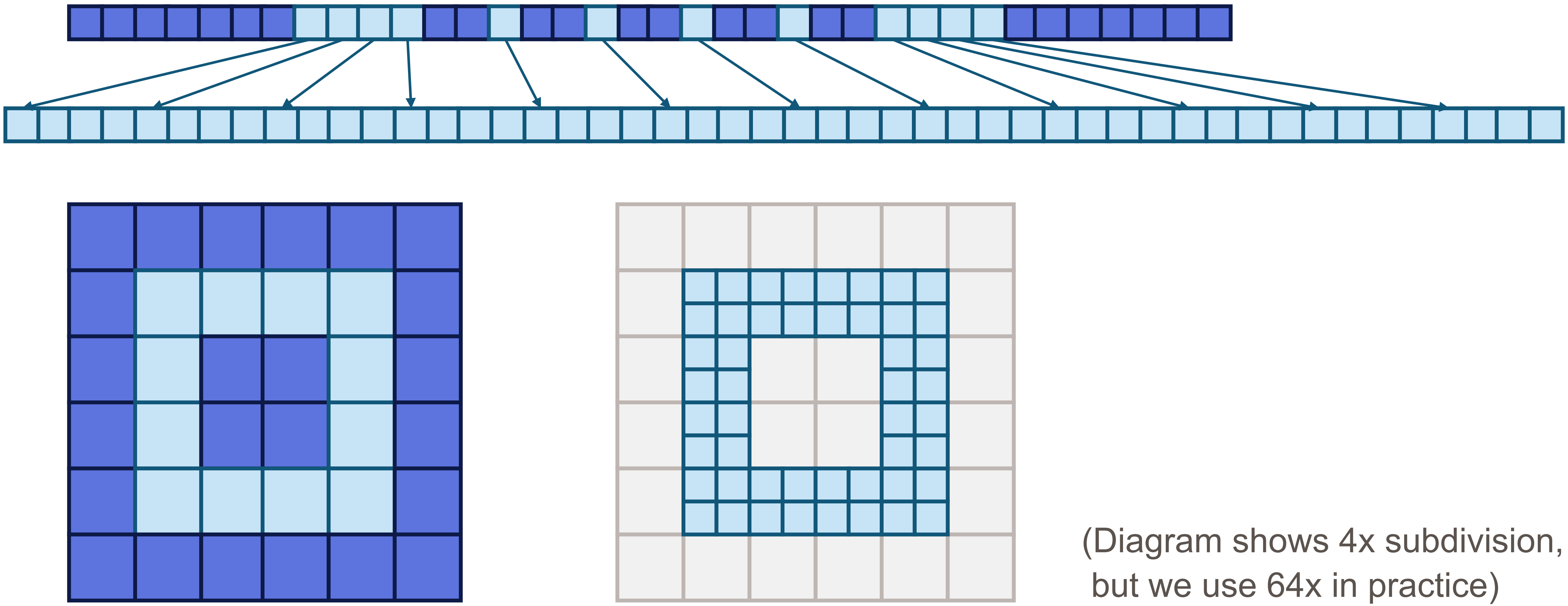
- Largest tiles are **densely packed**
  - $(N / 64)^D$ tiles, where N is image size and D is dimension
- Each tile contains three values
  - Tile position (packed 32-bit int, with 10 bits each for x/y/z)
  - Index of specialized tape for this tile (-1 by default)
  - Index of subdivided region for this tile (-1 by default)

MEMORY LAYOUT



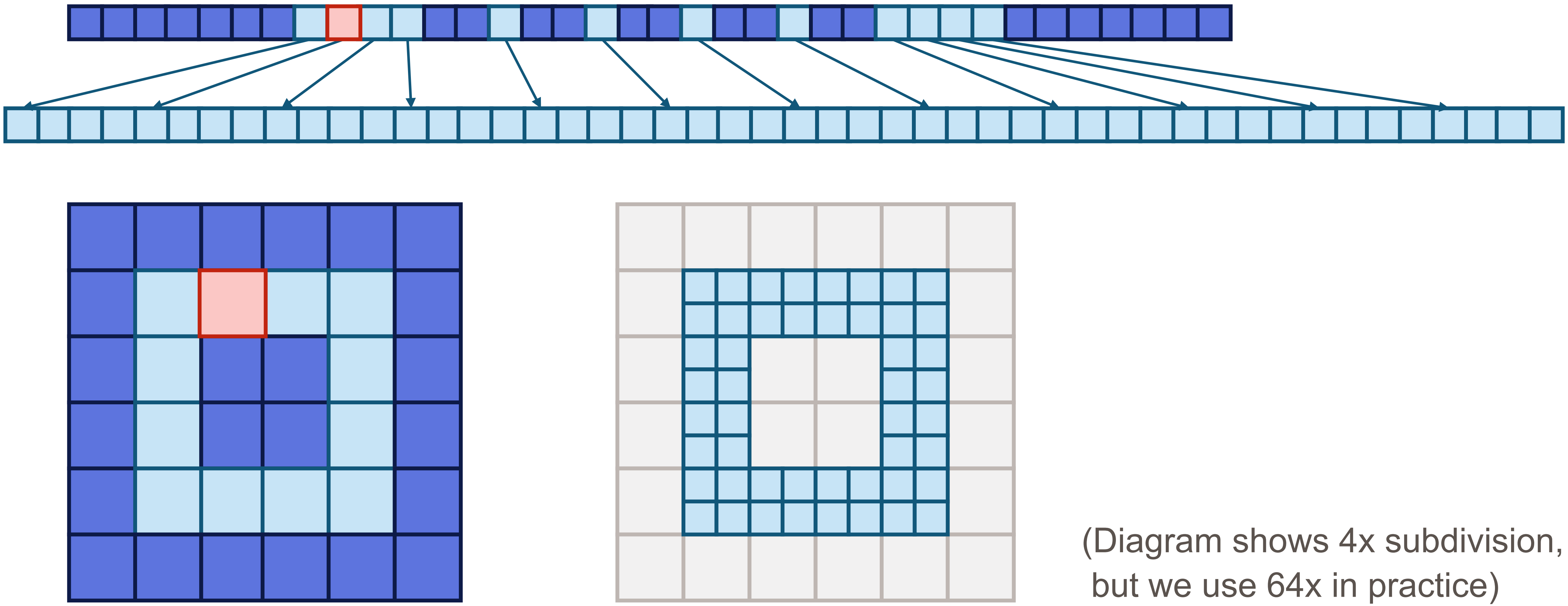
- Interval evaluation picks out ambiguous tiles
- These tiles will be subdivided and evaluated again
- Subdivided tiles are **sparsely distributed** in space but **densely packed** in GPU RAM

MEMORY LAYOUT



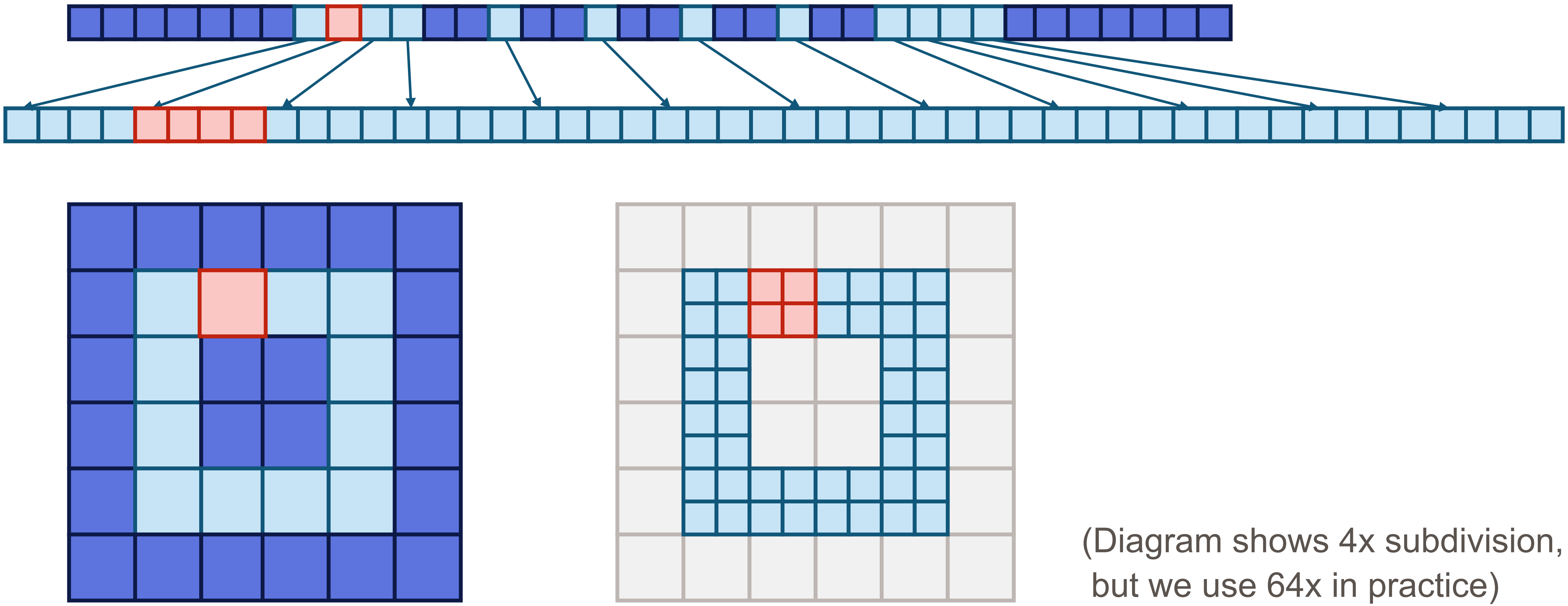
(Diagram shows 4x subdivision, but we use 64x in practice)

STORING THE HIERARCHY



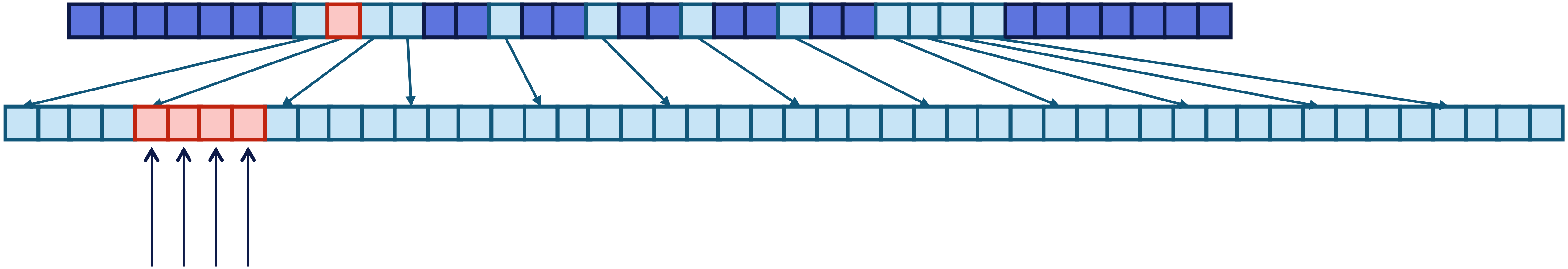
(Diagram shows 4x subdivision, but we use 64x in practice)

MEMORY LAYOUT



(Diagram shows 4x subdivision, but we use 64x in practice)

MEMORY LAYOUT



64 subtiles = 64 threads = 2 warps

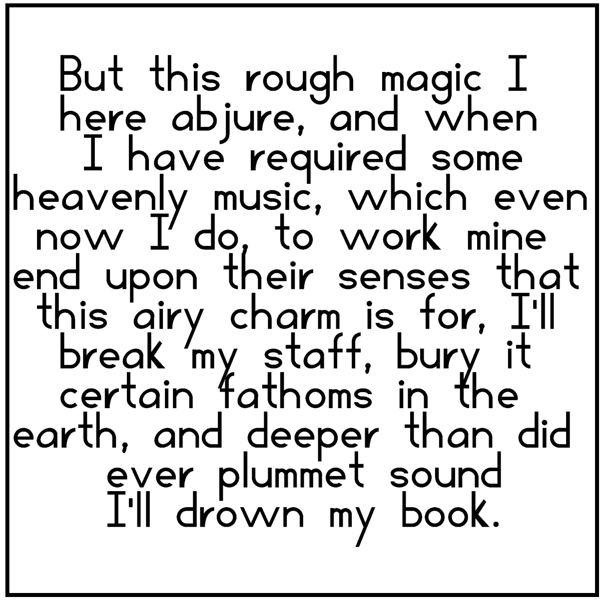
All threads within these warps use the **same shortened tape** (from parent tile), to minimize divergence

BENCHMARKING MACHINES

GeForce GT 750M
2013 MacBook Pro

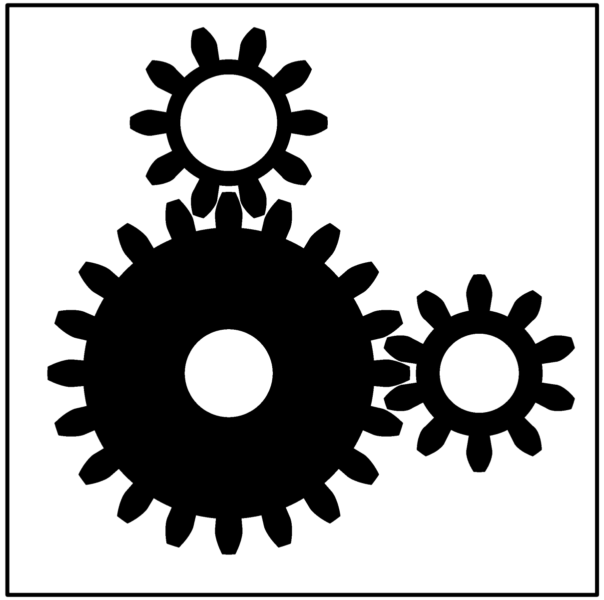
GTX 1080 Ti
2017 VR/ML workstation

Tesla V100
AWS `p3.2xlarge`



Text benchmark

- 6056 clauses (2354 min/max)
- Shows off CSG and tape pruning performance



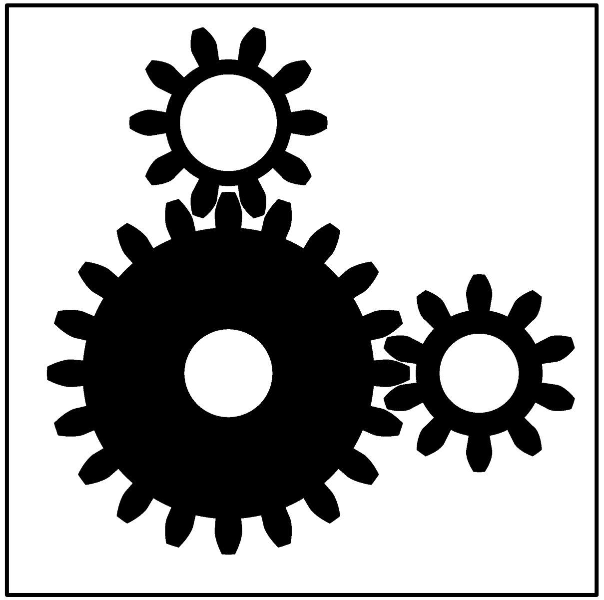
2D Gears

- 1735 clauses (374 min/max)
- Curves are mathematically correct involutes, which requires transcendental functions

But this rough magic I
here abjure, and when
I have required some
heavenly music, which even
now I do, to work mine
end upon their senses that
this airy charm is for, I'll
break my staff, bury it
certain fathoms in the
earth, and deeper than did
ever plummet sound
I'll drown my book.

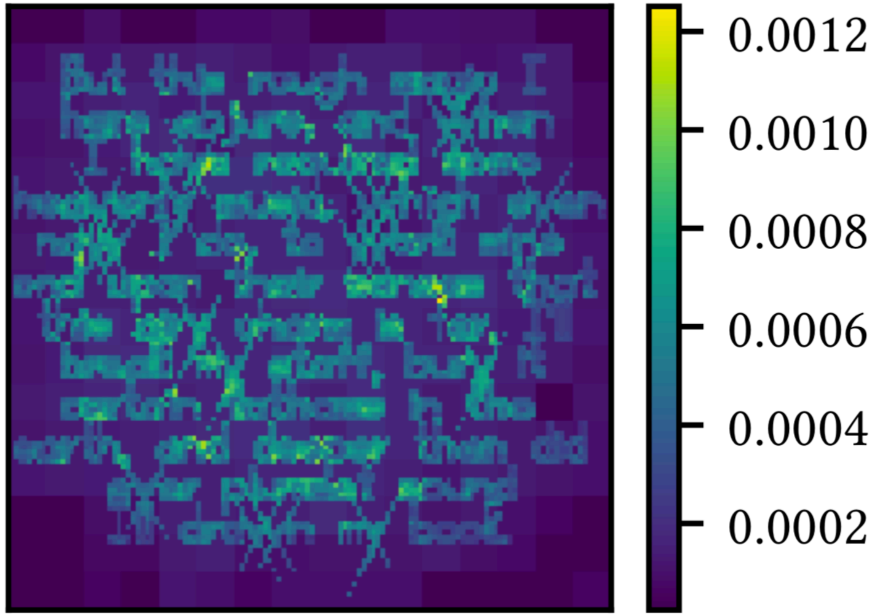
Frame time (ms)

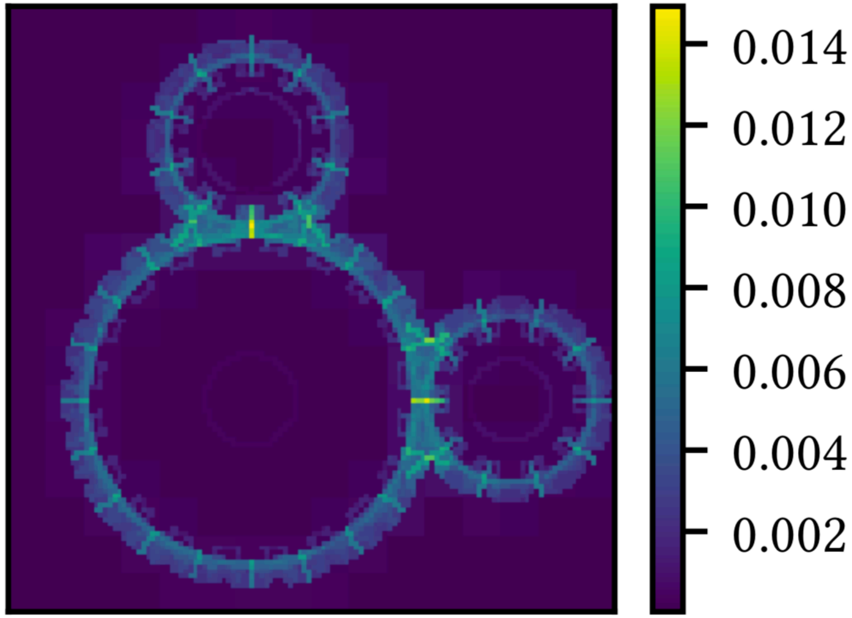
Size	GT 750M	GTX 1080 Ti	Tesla V100
256^2	17.5	8.3	5.2
512^2	14.8	6.8	4.2
1024^2	16.5	6.5	3.9
2048^2	20.7	6.6	3.9
3072^2	27.1	6.9	3.9
4096^2	35.9	7.4	4.1



Frame time (ms)

Size	GT 750M	GTX 1080 Ti	Tesla V100
256^2	9.2	4.0	2.8
512^2	9.3	3.7	2.5
1024^2	12.1	3.4	2.2
2048^2	17.3	3.4	2.2
3072^2	23.4	3.7	2.3
4096^2	30.6	4.0	2.4

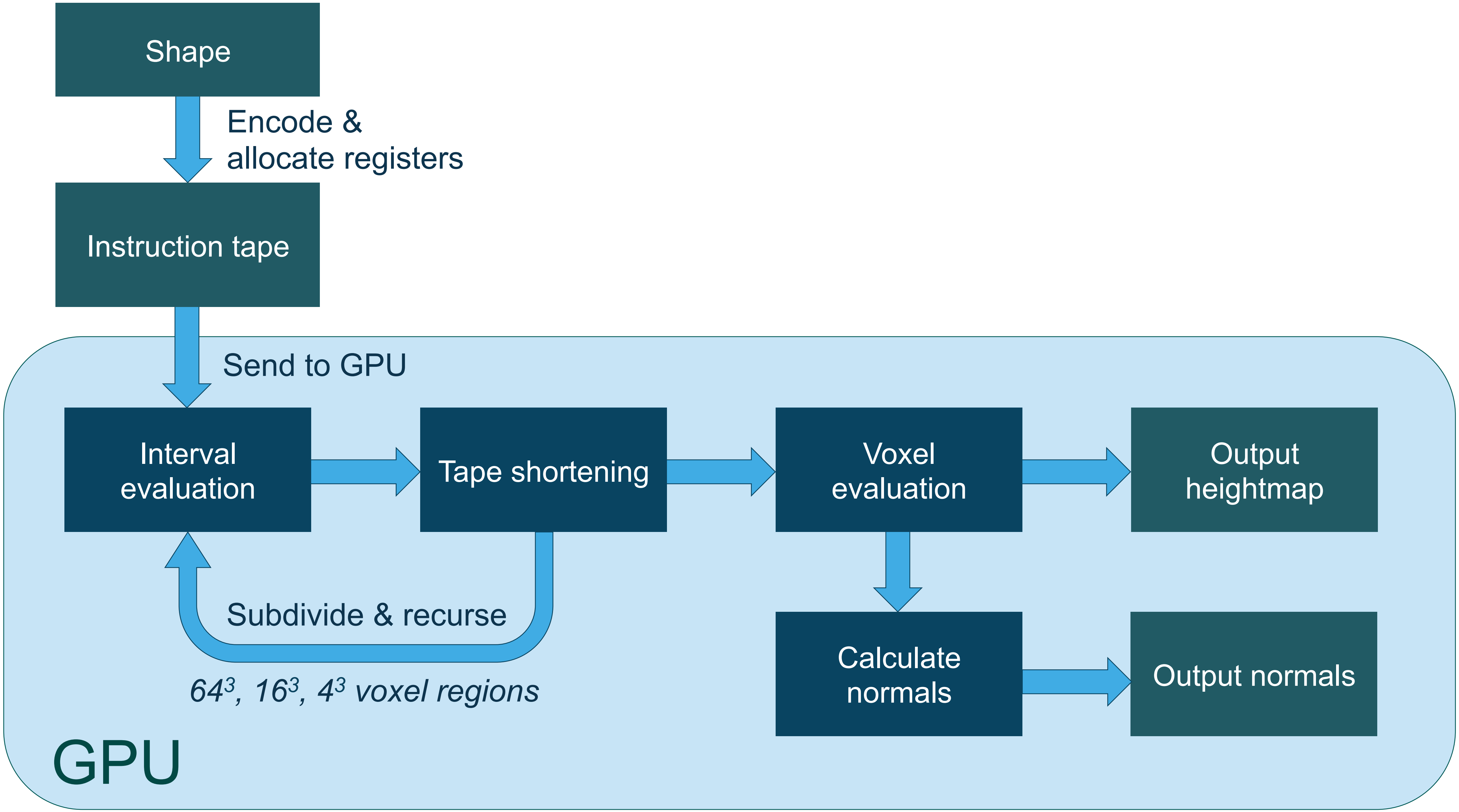




Metric: Normalized + amortized work

- A score of 1 means that the tape is fully walked once for that pixel
- Interval evaluation work is amortized across all pixels in the interval
- Less than 1 full-tape evaluation per pixel!

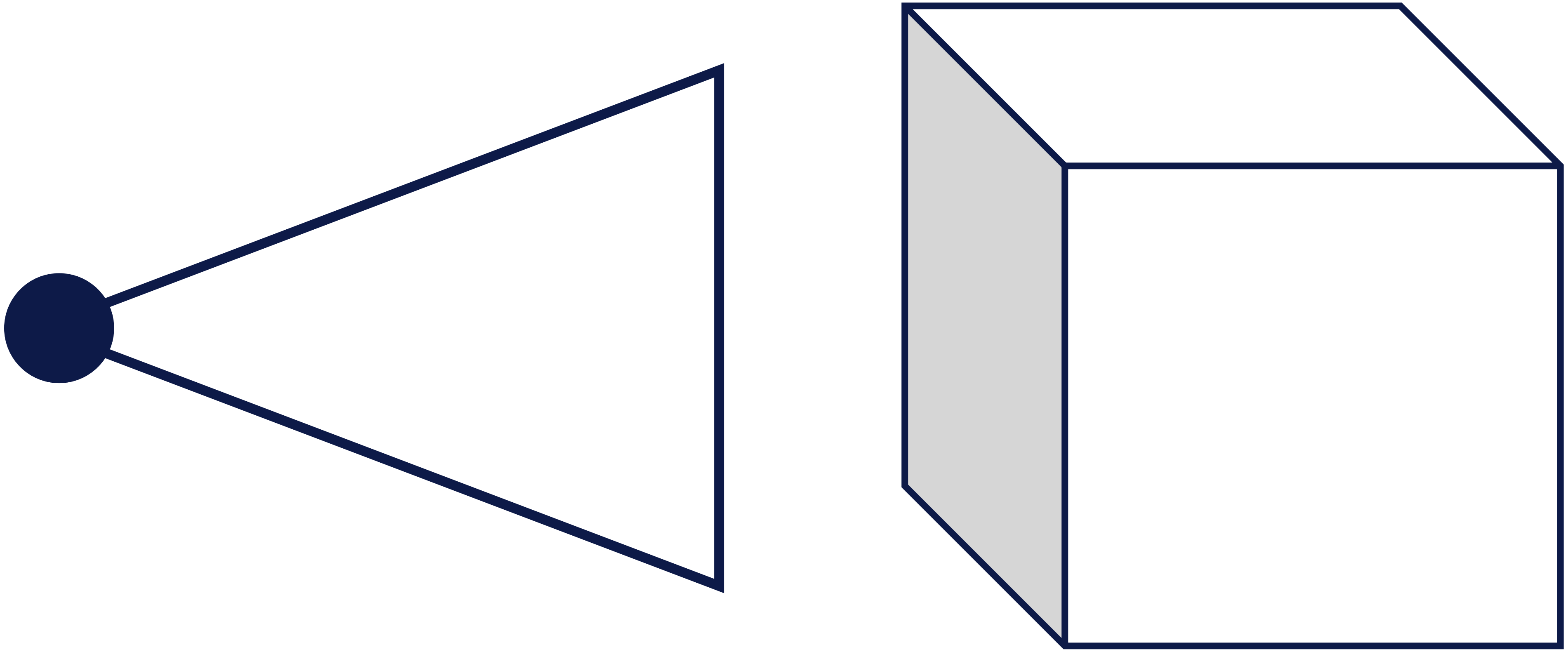
RENDERING IN 3D


Shape
Encode &
allocate registers
Instruction tape
Send to GPU
Interval evaluation
Tape shortening
Voxel evaluation
Output heightmap
Subdivide & recurse
64³, 16³, 4³ voxel regions
Calculate normals
Output normals
GPU

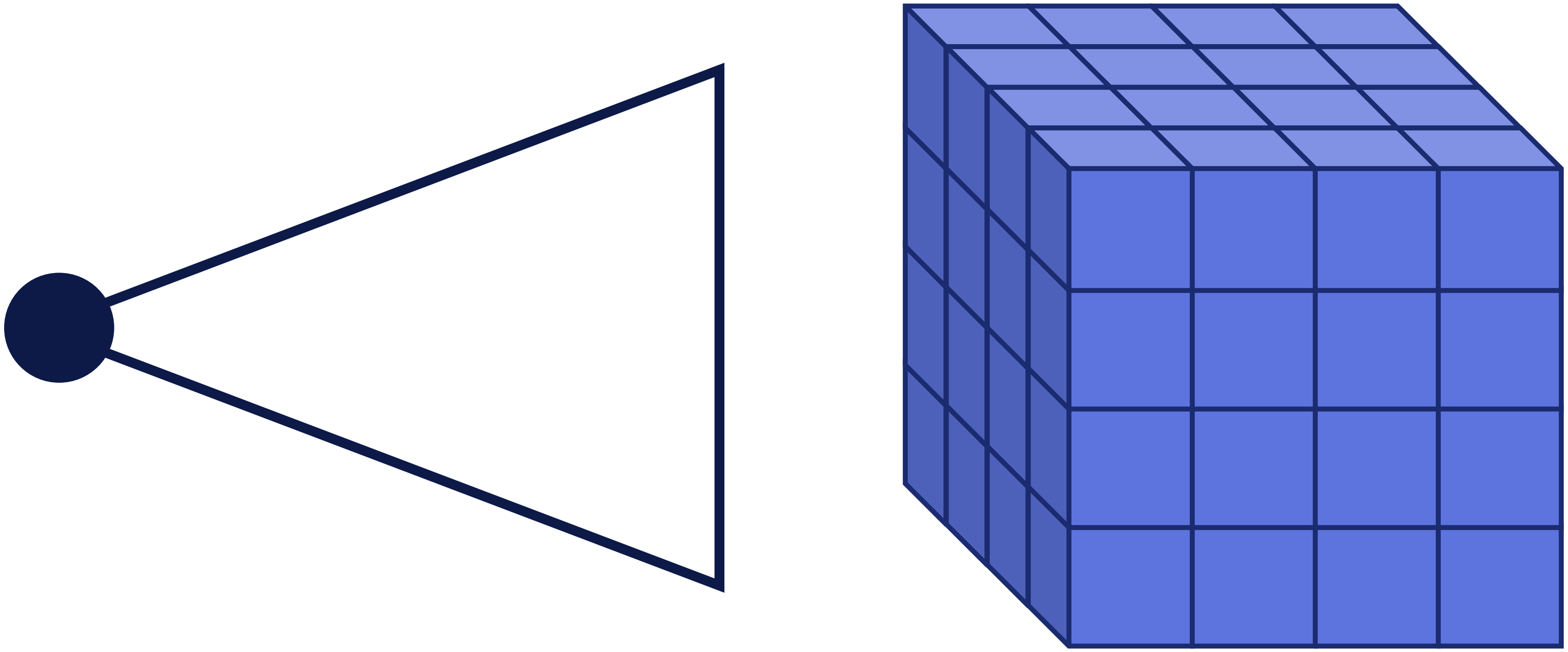
3D RENDERING PROCEDURE

- Interval evaluation of 64x64x64-voxel **tiles**
  - Split ambiguous tiles into 64 **subtiles** (16x16x16-voxels) each
- Interval evaluation of **subtiles**
  - Split ambiguous subtiles into 64 **microtiles** (4x4x4-voxels) each
- Interval evaluation of **microtiles**
- Per-**voxel** evaluations of remaining ambiguous **microtiles**
- Per-**pixel** evaluation, using automatic differentiation to find normals
- *Optional:* post-processing depth + normal buffer to generate final image

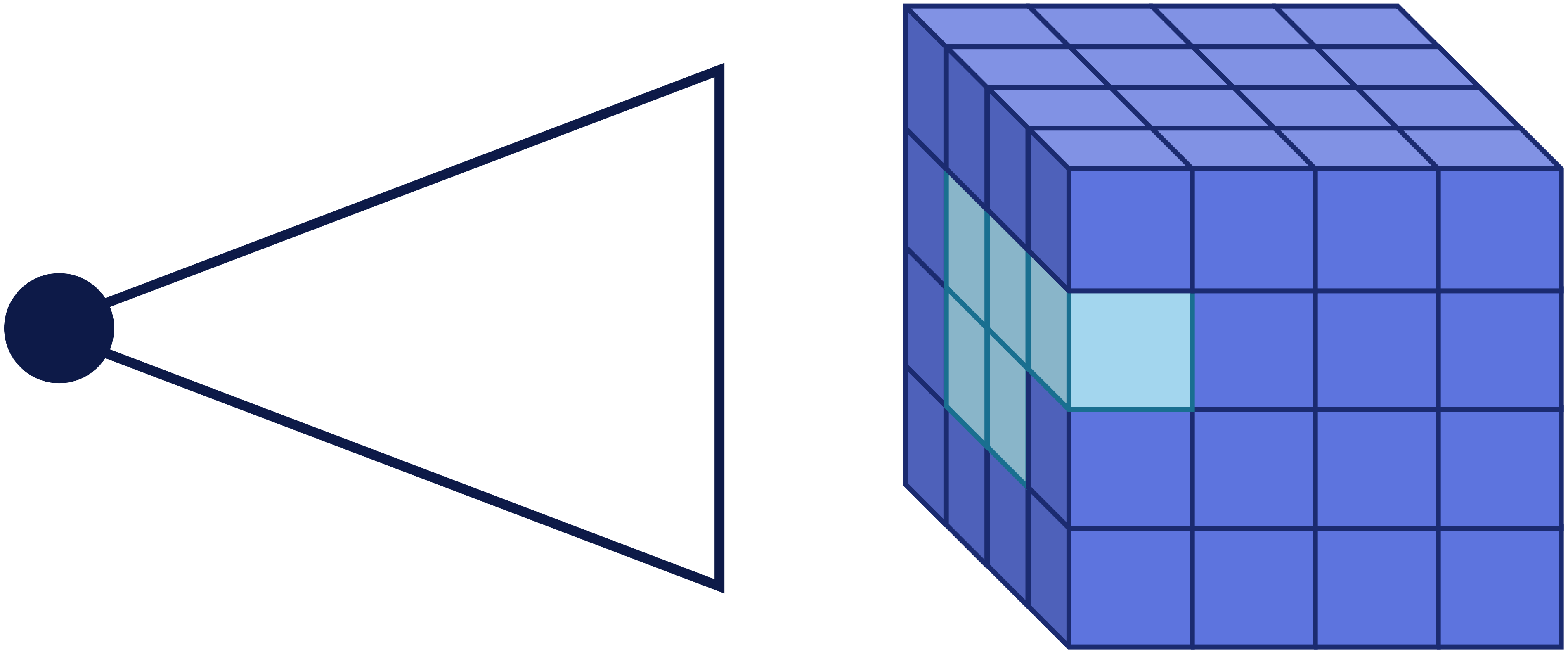
RENDERING A HEIGHTMAP



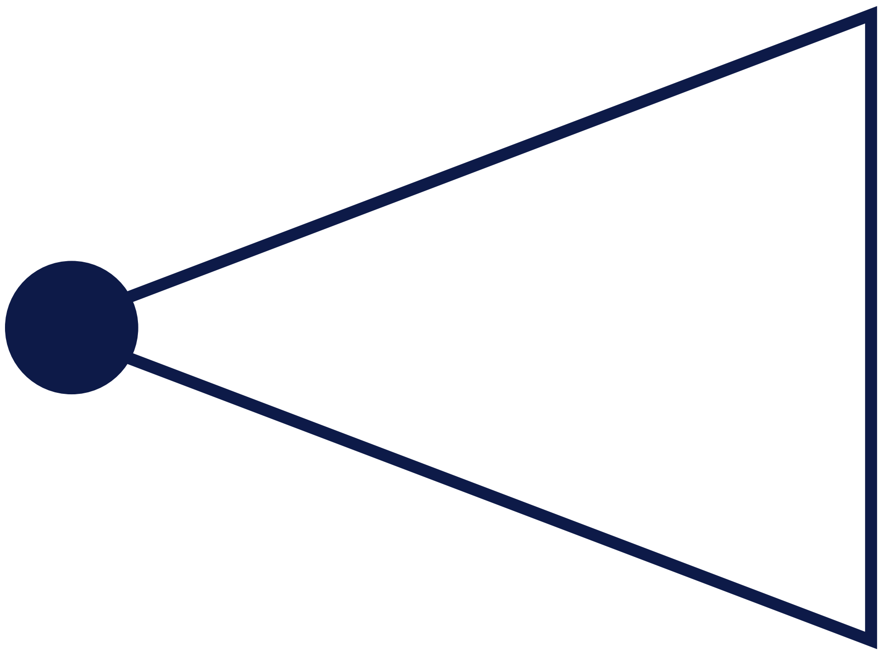
RENDERING A HEIGHTMAP

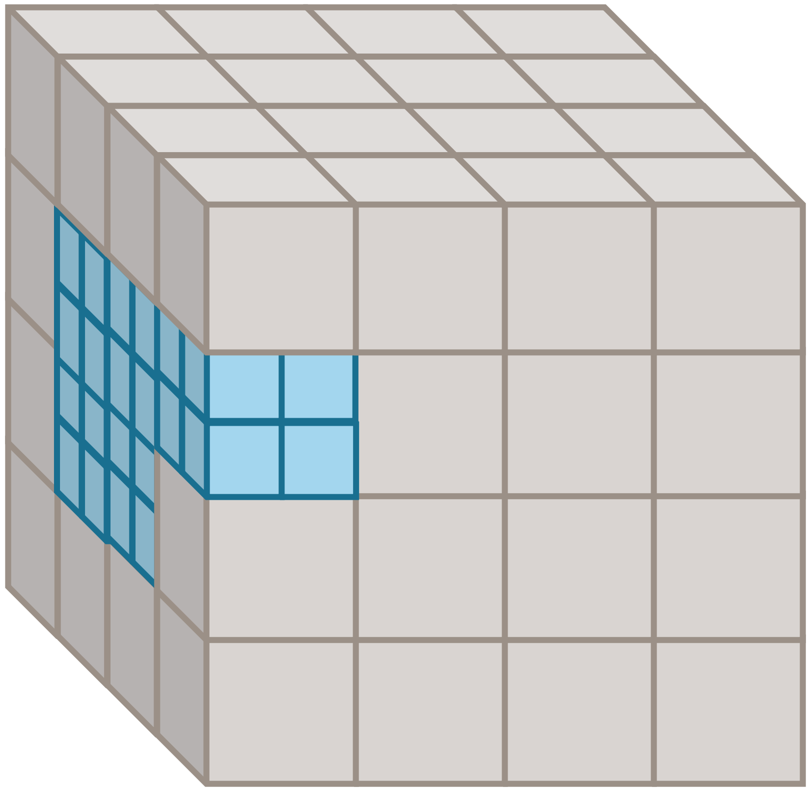


RENDERING A HEIGHTMAP

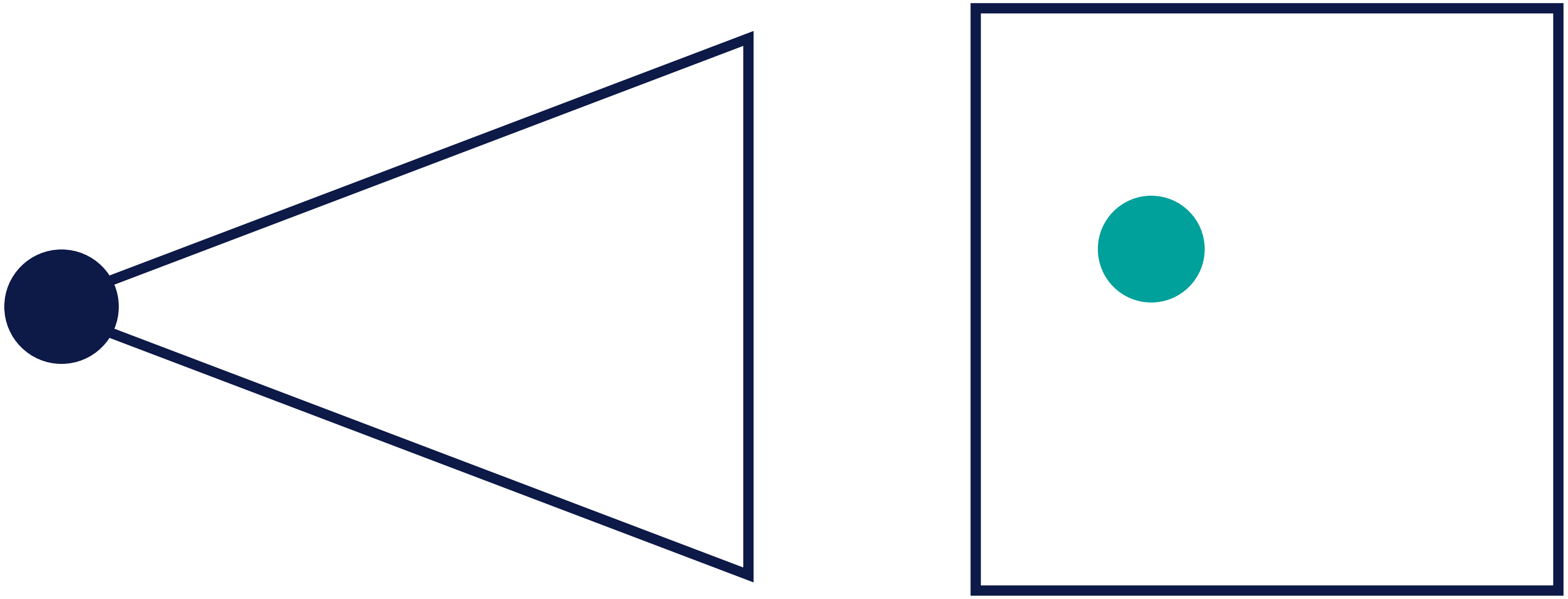


RENDERING A HEIGHTMAP

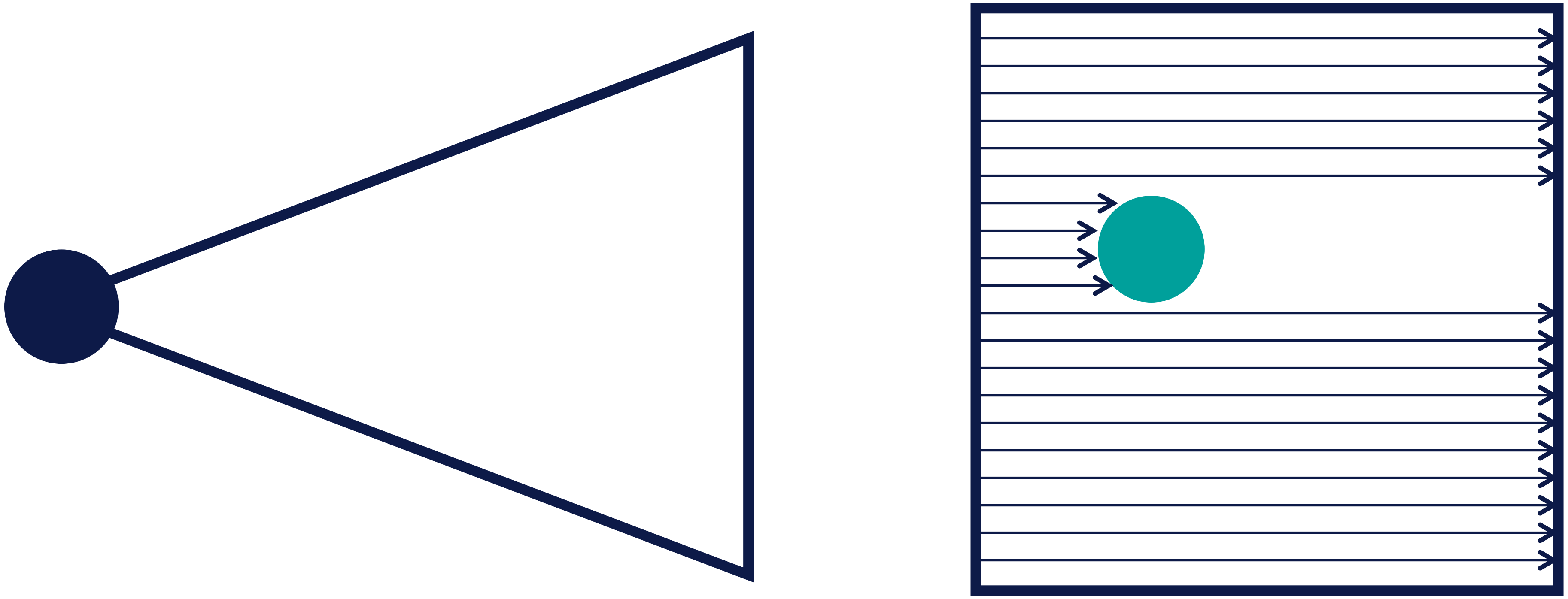




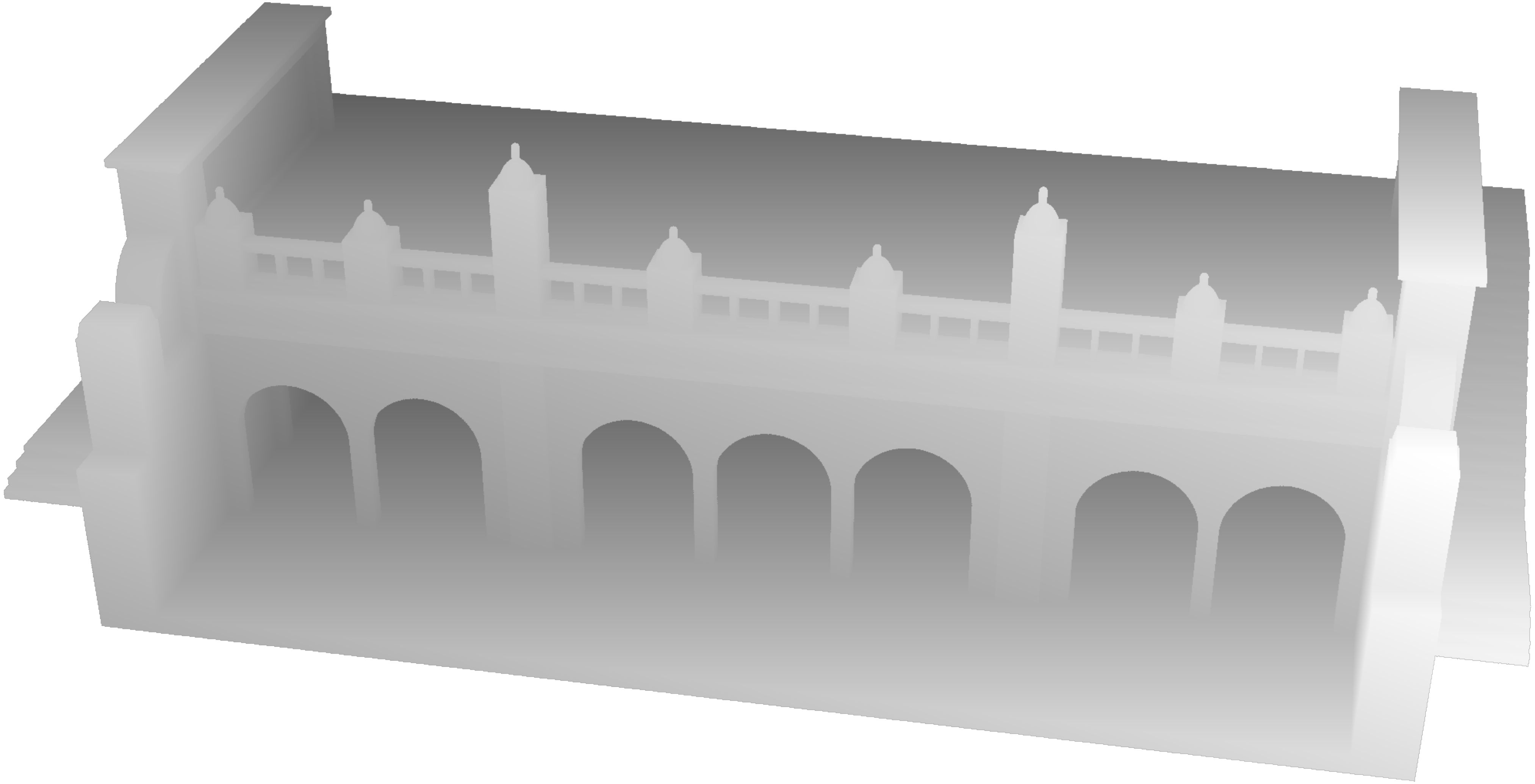
ORTHOGRAPHIC HEIGHTMAP



ORTHOGRAPHIC HEIGHTMAP



HEIGHTMAP OUTPUT

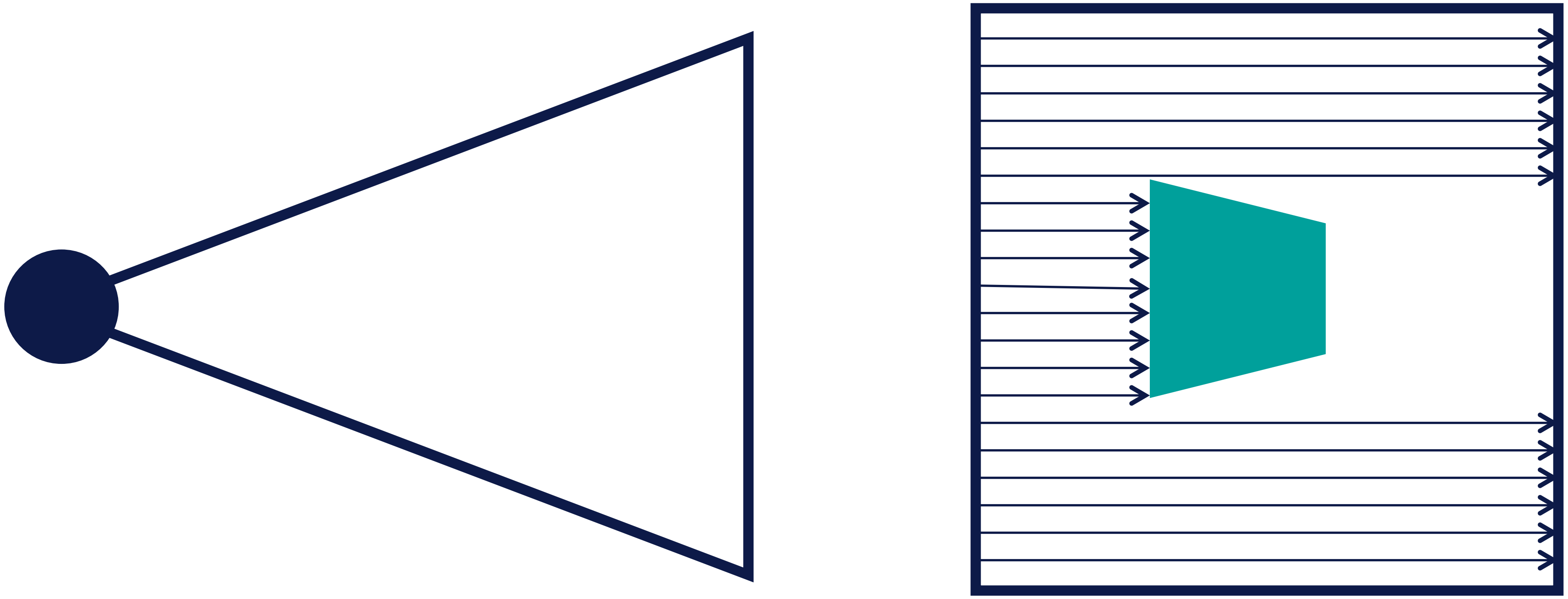


USER-PROVIDED MATRIX

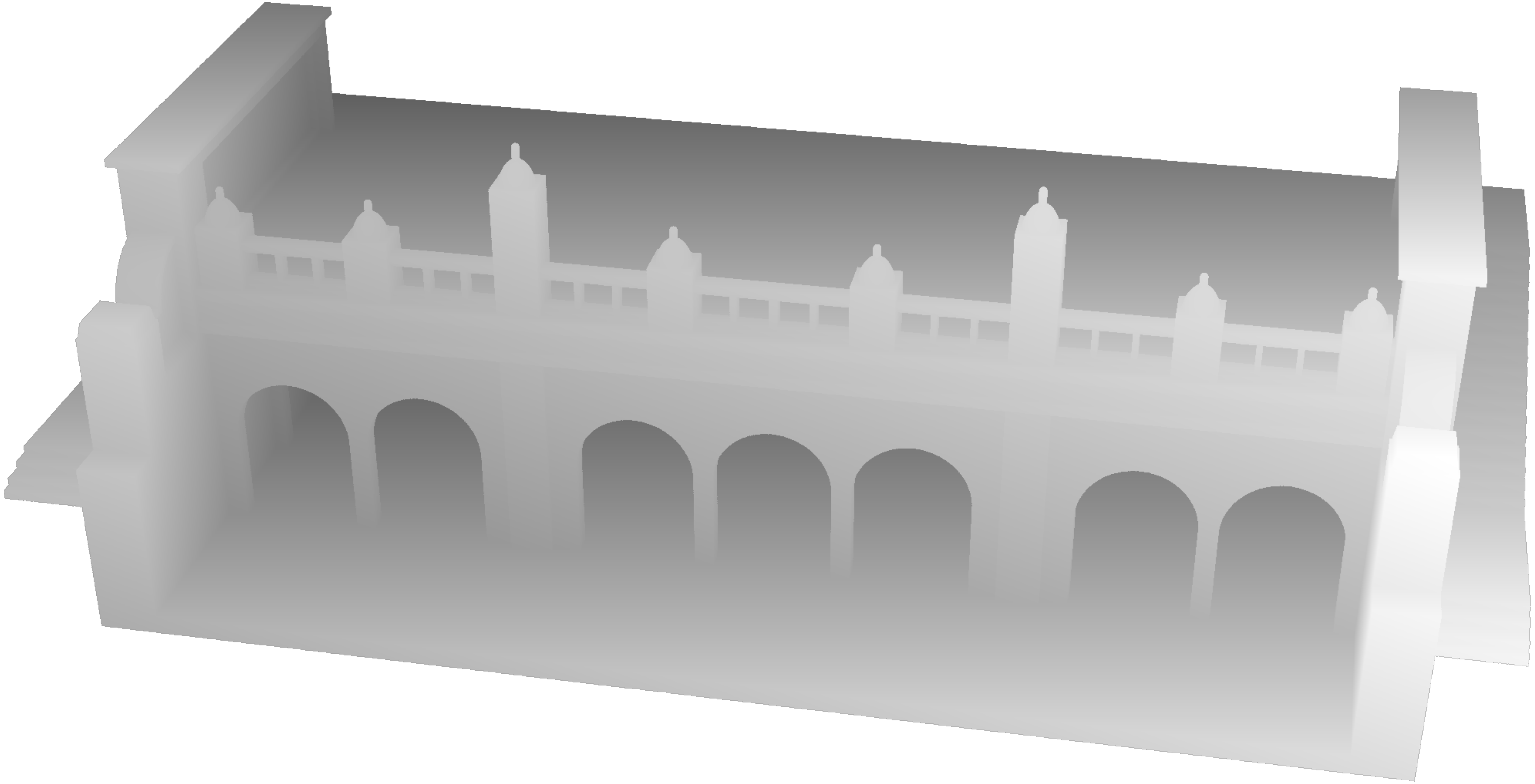
opcode	lhs	rhs	out
X	–	–	slot 0
Y	–	–	slot 1
SQUARE	slot 0	–	slot 0
SQUARE	slot 1	–	slot 1
ADD	slot 0	slot 1	slot 1
SQRT	slot 1	–	slot 1
SUB	slot 1	1.0f	slot 0
SUB	0.5f	slot 1	slot 1
MAX	slot 0	slot 1	slot 1

- `XYZ` opcodes return coordinates that are transformed by a 4x4 matrix
- OpenGL-style homogenous coordinates (allows for perspective)
- Also used to implementing the camera
- Faster than transforming the model and uploading a new tape

USER-PROVIDED MATRIX

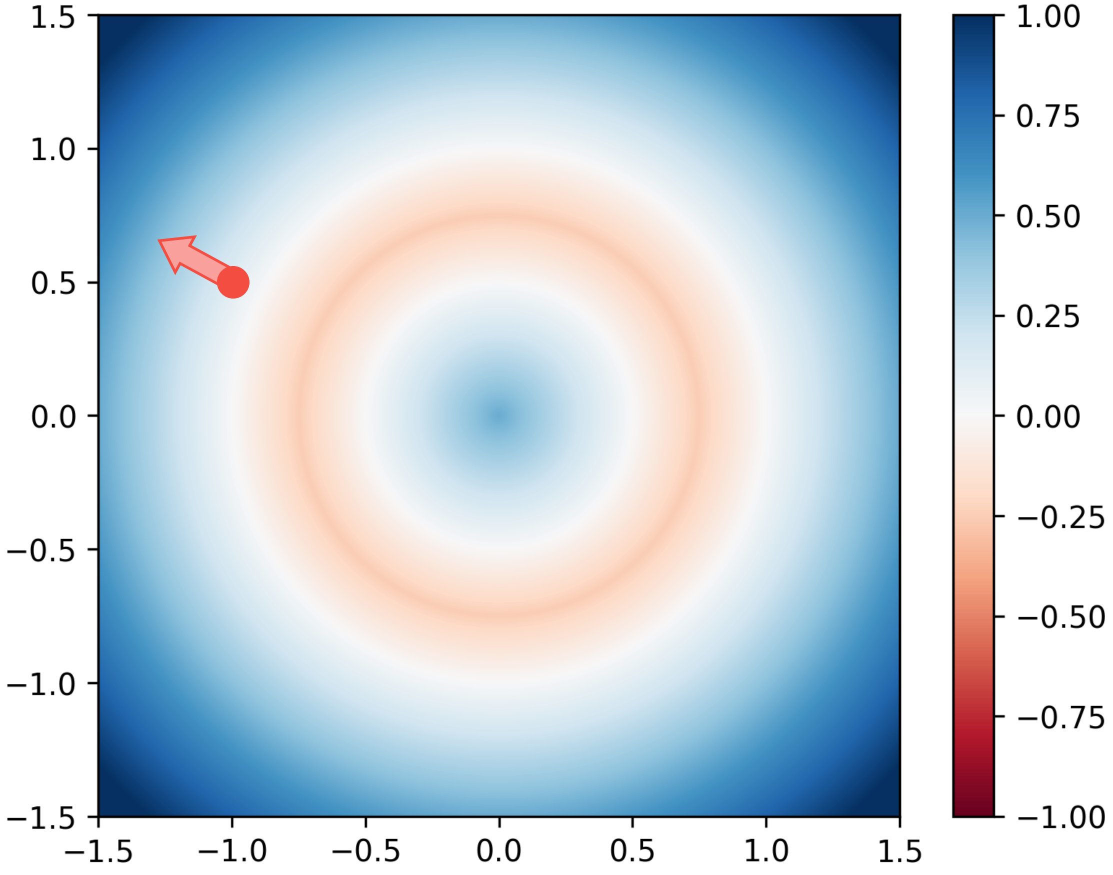


HEIGHTMAP OUTPUT



CALCULATING NORMALS

- The **partial derivatives**
  `df/dx, df/dy, df/dz`
  approximate the normal at points close to the model surface
- **Forward-mode automatic differentiation** can be implemented by interpreting the tape with **value + partial derivatives**



2D AUTOMATIC DIFFERENTIATION

opcode	lhs	rhs	out
X	–	–	-1.0, (1, 0)
Y	–	–	0.5, (0, 1)
SQUARE		–	
SQUARE		–	
ADD			
SQRT		–	
SUB		1.0f	
SUB	0.5f		
MAX			

2D AUTOMATIC DIFFERENTIATION

opcode	lhs	rhs	out
X	–	–	-1.0, (1, 0)
Y	–	–	0.5, (0, 1)
SQUARE	-1.0, (1, 0)	–	
SQUARE		–	
ADD			
SQRT		–	
SUB		1.0f	
SUB	0.5f		
MAX			

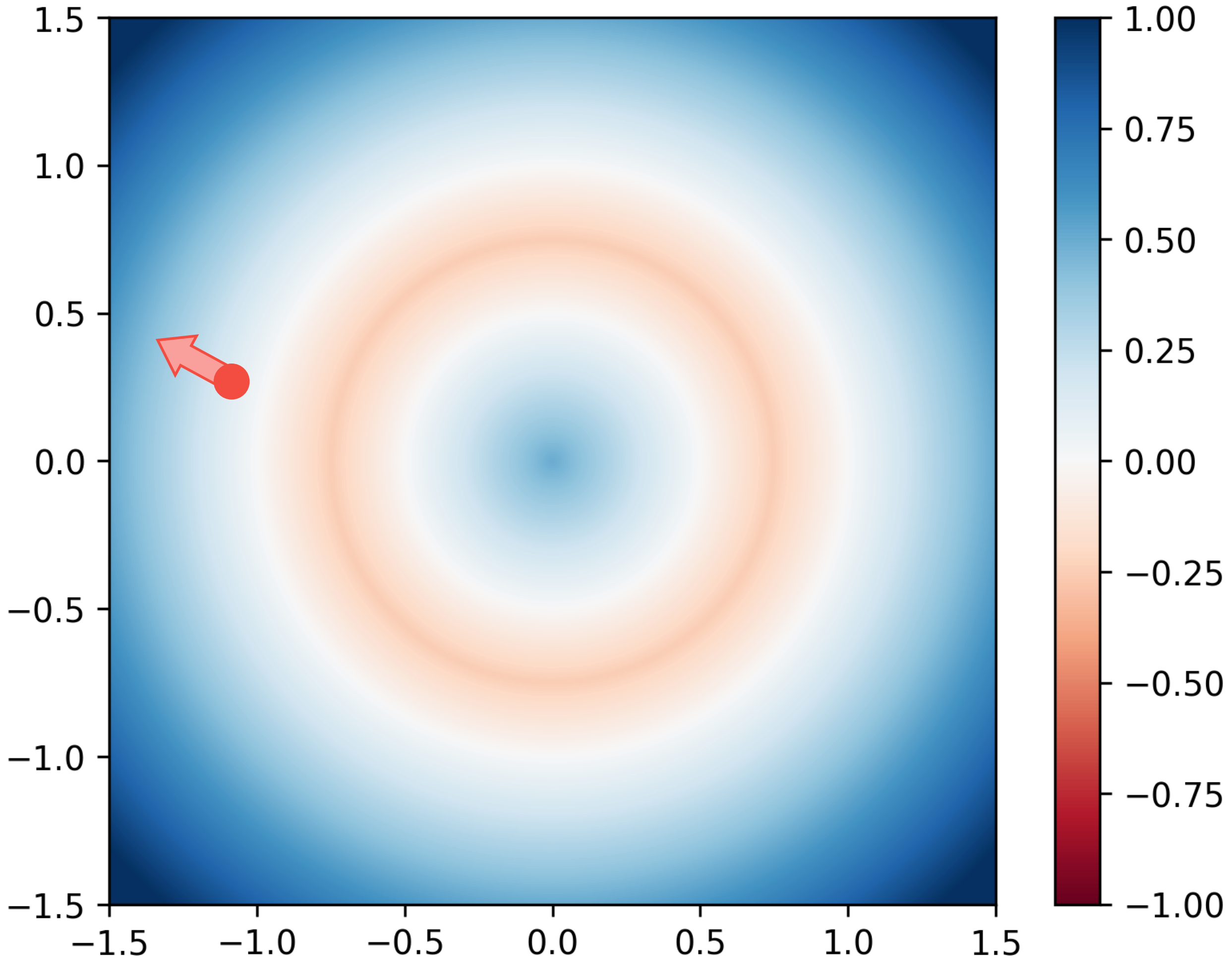
2D AUTOMATIC DIFFERENTIATION

opcode	lhs	rhs	out
X	–	–	-1.0, (1, 0)
Y	–	–	0.5, (0, 1)
SQUARE	-1.0, (1, 0)	–	1.0, (-2, 0)
SQUARE		–	
ADD			
SQRT		–	
SUB		1.0f	
SUB	0.5f		
MAX			

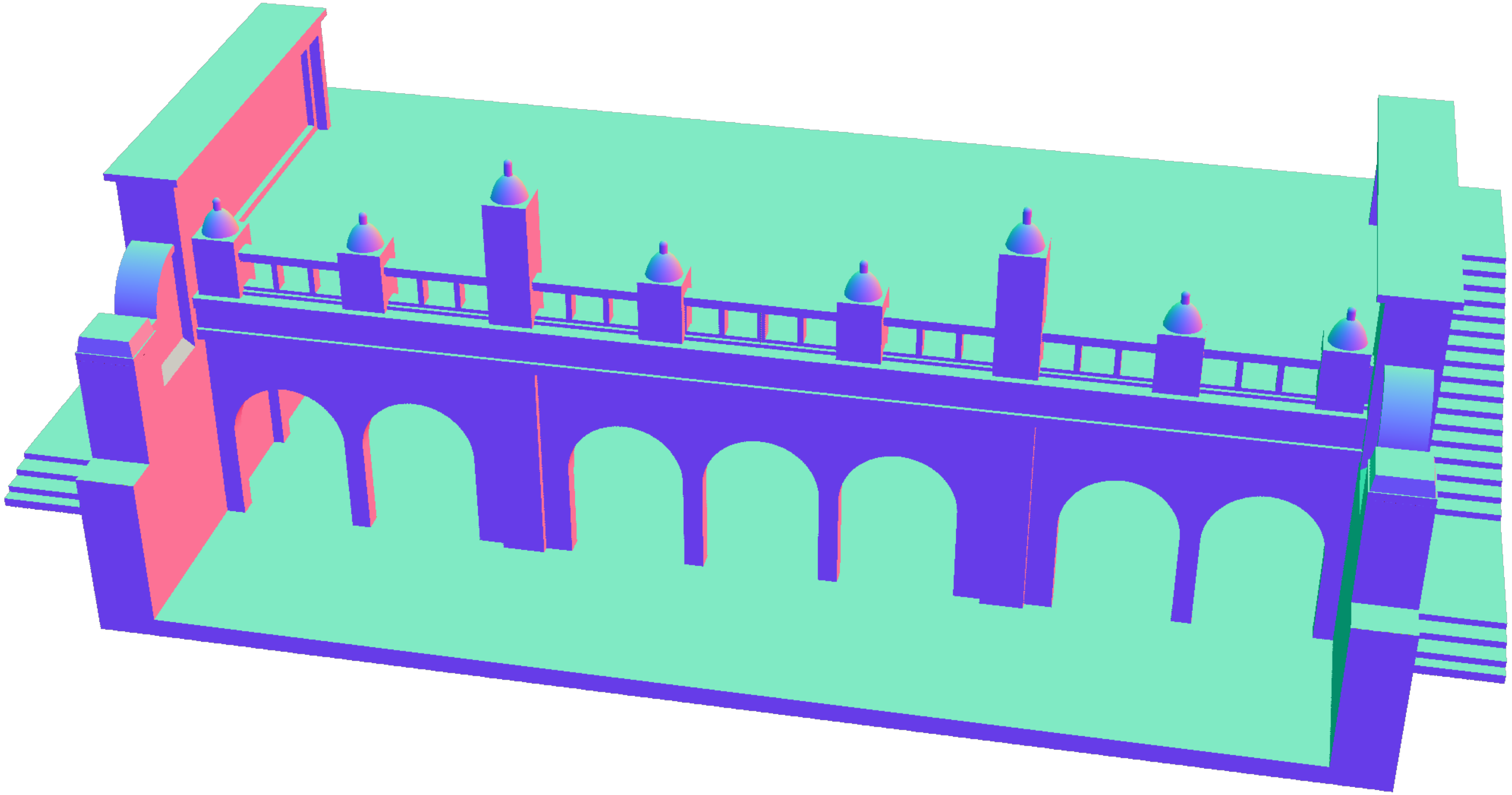
2D AUTOMATIC DIFFERENTIATION

opcode	lhs	rhs	out
X	–	–	-1.0, (1, 0)
Y	–	–	0.5, (0, 1)
SQUARE	-1.0, (1, 0)	–	1.0, (-2, 0)
SQUARE	0.5, (0, 1)	–	0.25, (0, 1)
ADD	1.0, (-2, 0)	0.25, (0, 1)	1.25, (-2, 1)
SQRT	1.25, (-2, 1)	–	1.12, (-0.89, 0.45)
SUB	1.12, (-0.89, 0.45)	1.0f	0.12, (-0.89, 0.45)
SUB	0.5f	1.12, (-0.89, 0.45)	-0.62, (0.89, -0.45)
MAX	0.12, (-0.89, 0.45)	-0.62, (0.89, -0.45)	0.12, (-0.89, 0.45)

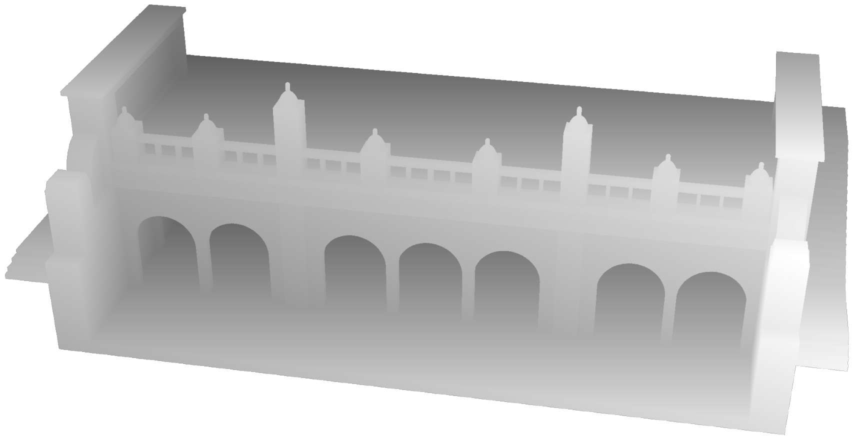
CALCULATING NORMALS

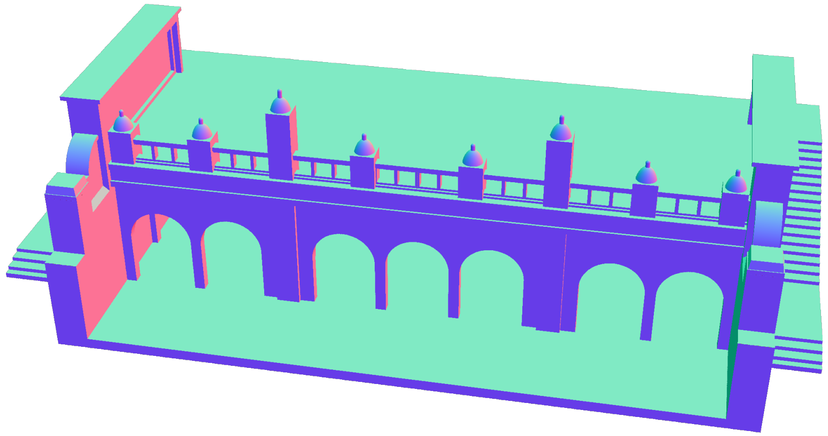


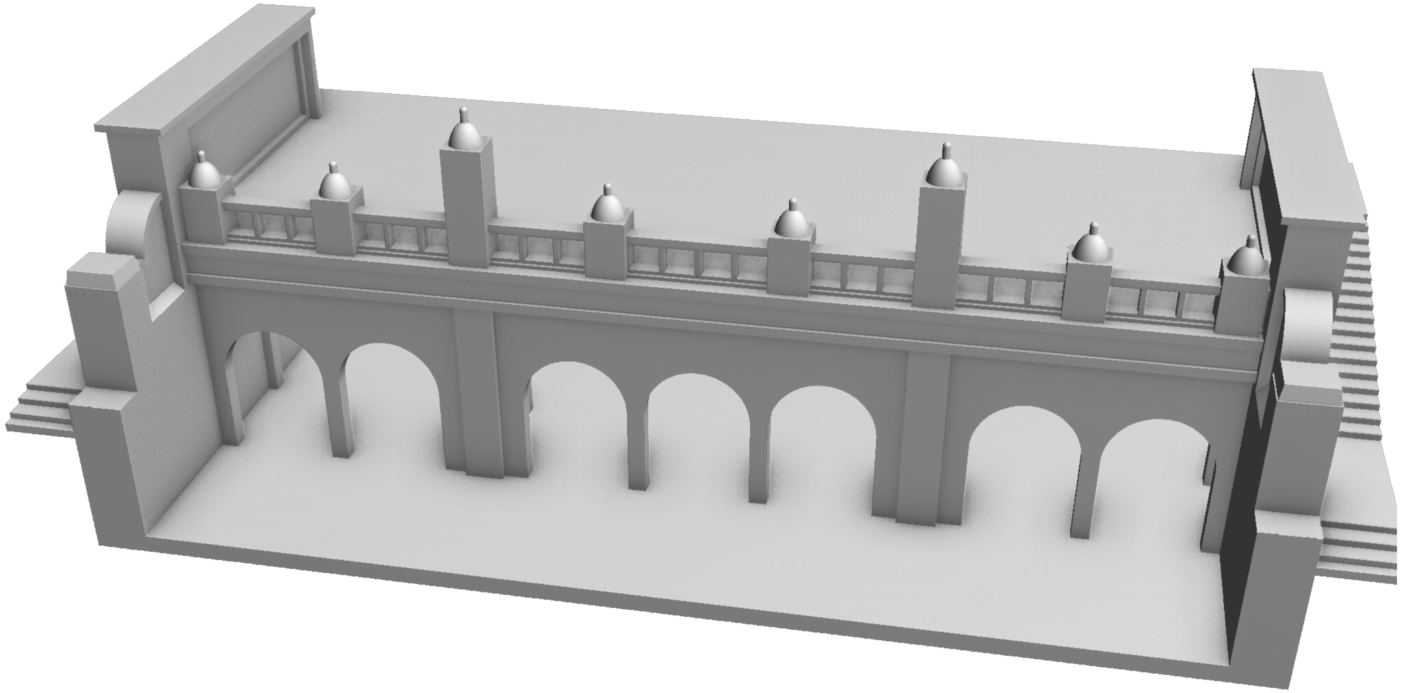
NORMALS OUTPUT



POST-PROCESSING







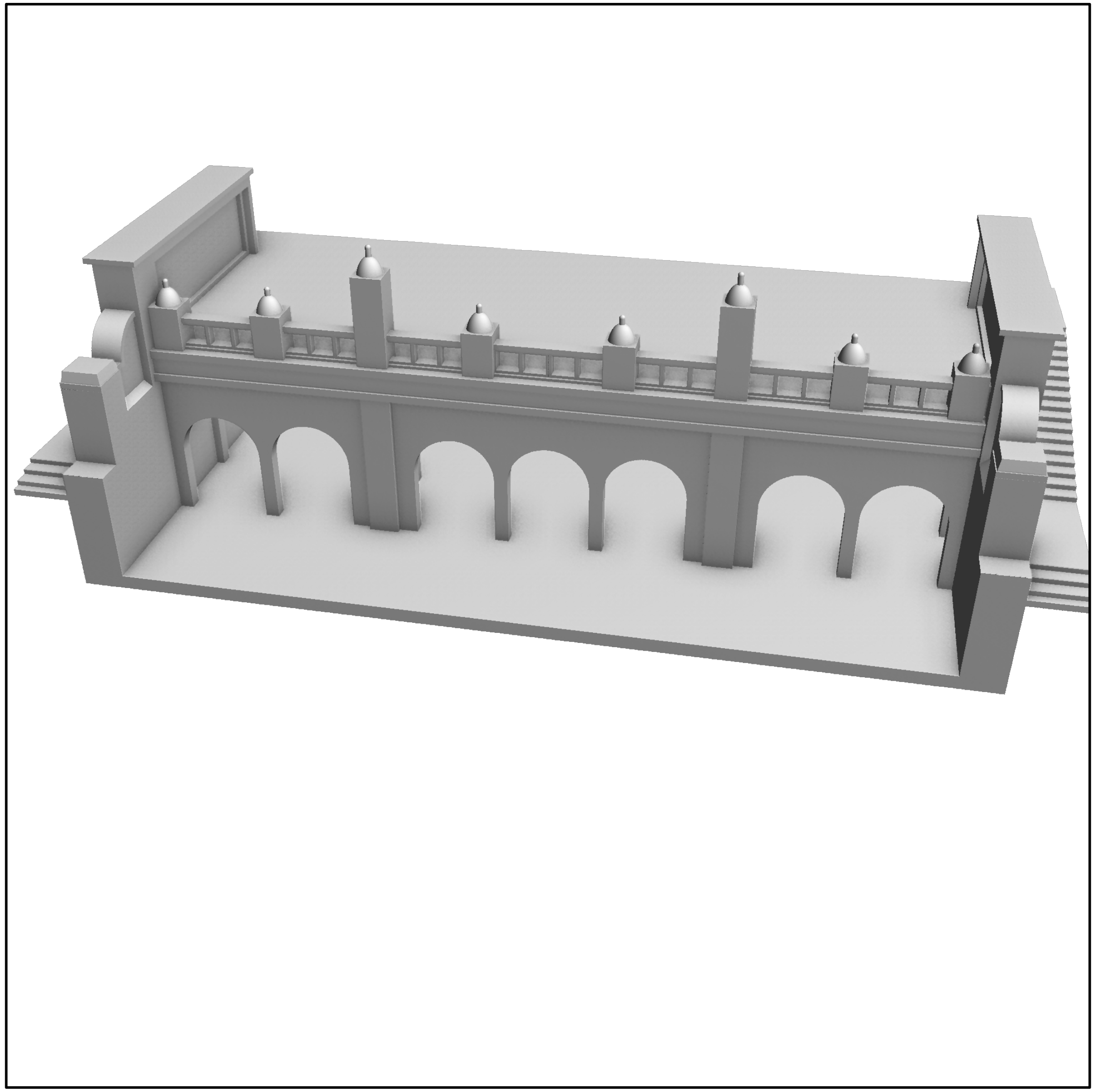
BENCHMARKING MACHINES

GeForce GT 750M
2013 MacBook Pro

GTX 1080 Ti
2017 VR/ML workstation

Tesla V100
AWS `p3.2xlarge`

ARCHITECTURE MODEL



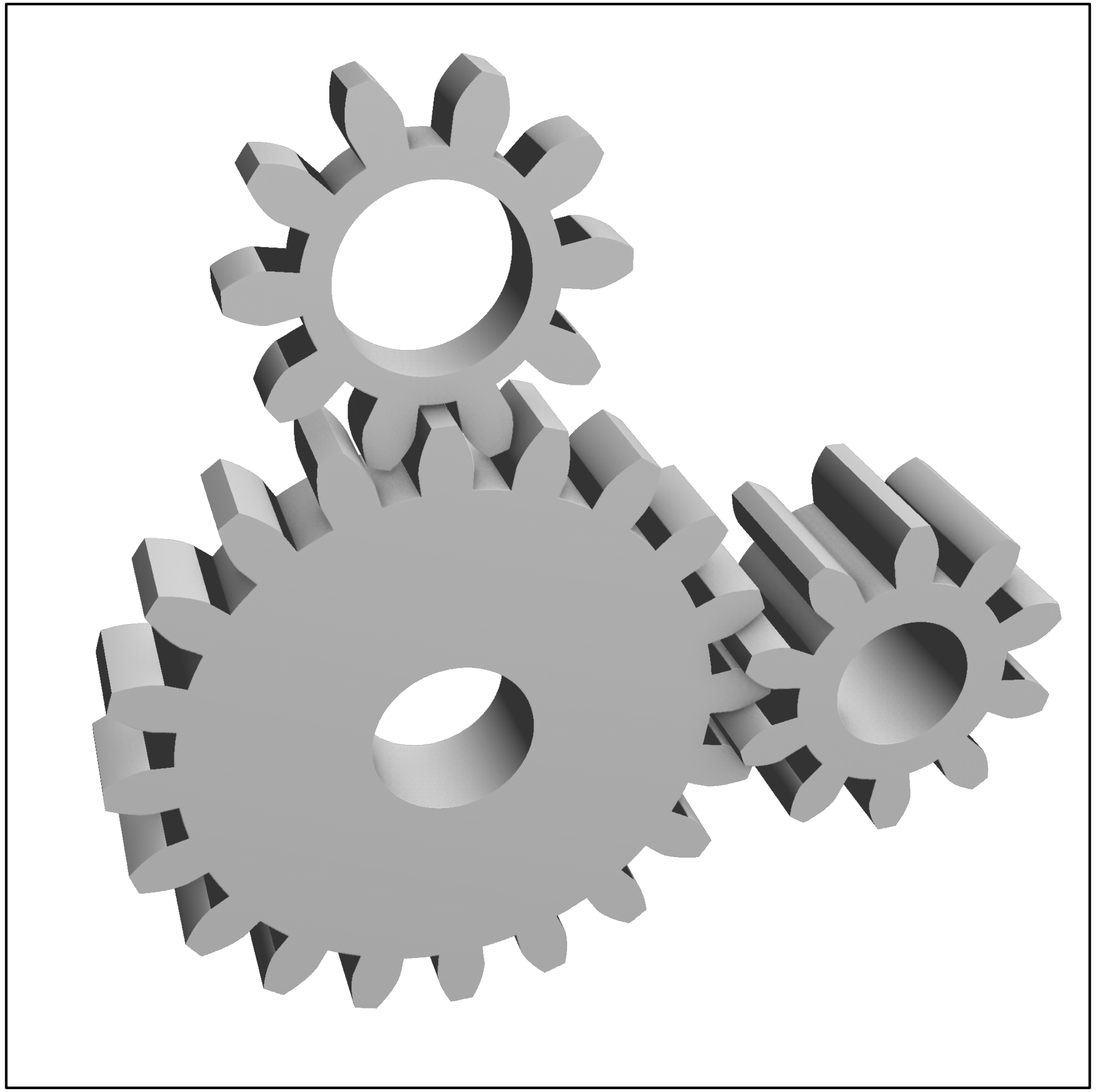
Frame time (ms)

Size	GT 750M	GTX 1080 Ti	Tesla V100
256^3	34.3	5.5	3.2
512^3	73.9	9.9	5.3
1024^3	189.9	22.6	12.2
1536^3	331.9	39.3	20.8
2048^3	510.7	60.6	31.9

- 961 clauses (465 min/max)
- CSG heavy model
- Best case for our algorithm

Based on a model by
Jennifer Keeter

GEARS MODEL



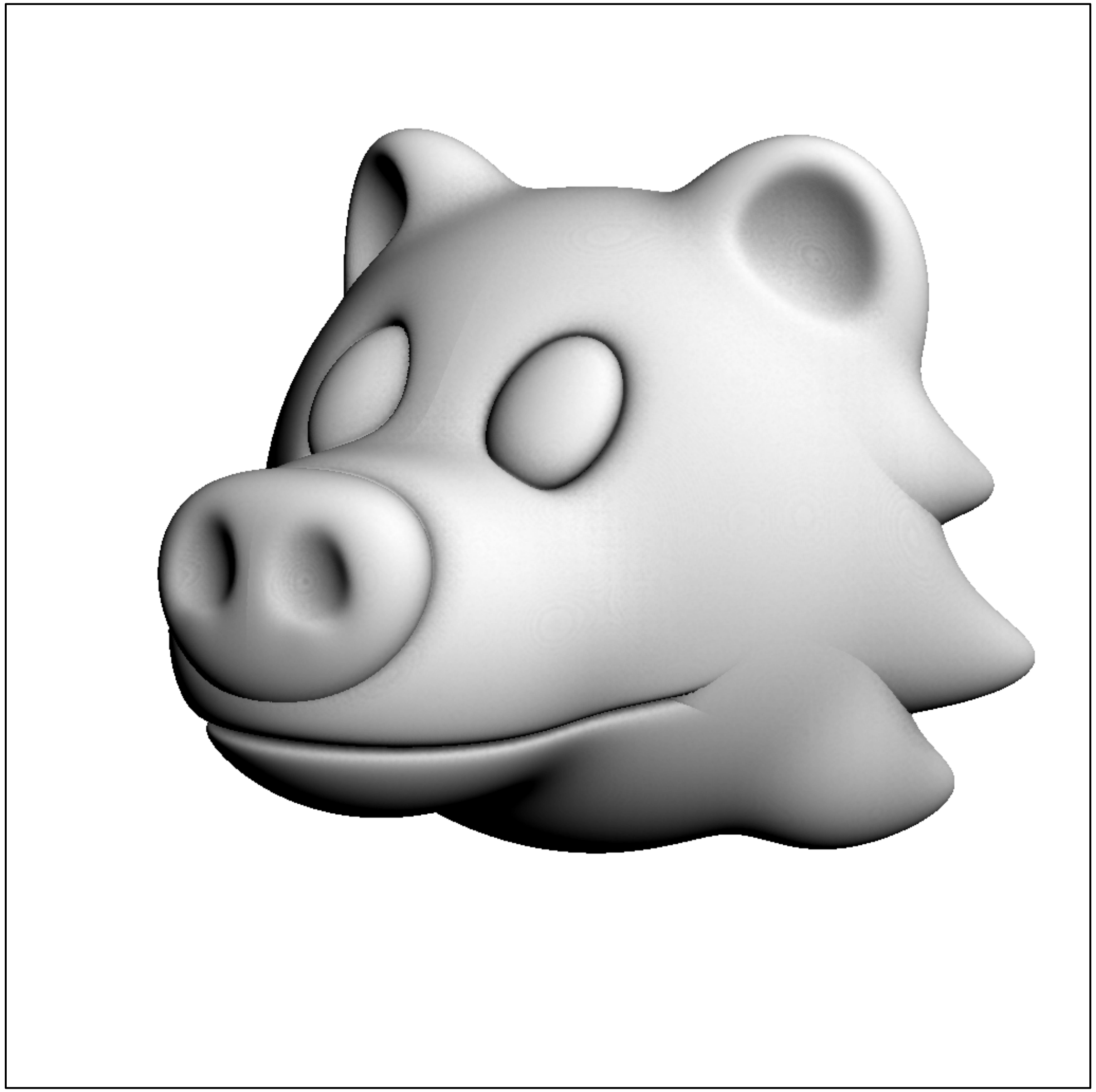
Frame time (ms)

Size	GT 750M	GTX 1080 Ti	Tesla V100
256^3	65.0	9.4	6.2
512^3	154.5	16.6	9.2
1024^3	426.2	40.3	23.1
1536^3	930.3	72.0	39.5
2048^3	–	115.4	62.0

- 1735 clauses (374 min/max)
- Medium amount of CSG
- Transcendental functions

Clever involute curve derivation by Peter Fedak

BEAR HEAD MODEL



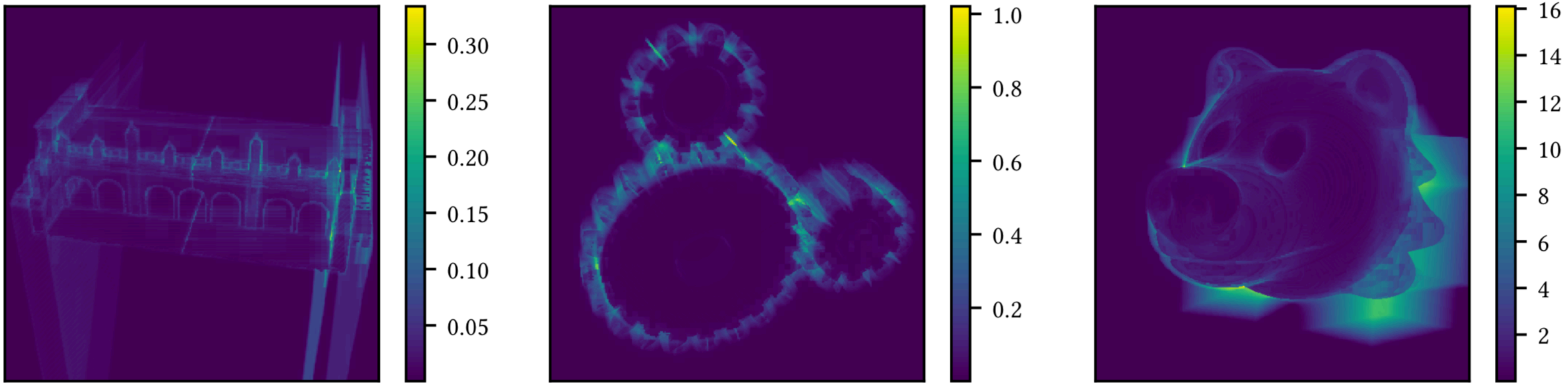
Frame time (ms)

Size	GT 750M	GTX 1080 Ti	Tesla V100
256^3	111.3	11.3	5.2
512^3	503.6	41.1	20.3
1024^3	2352.1	191.0	88.3
1536^3	–	504.2	228.3
2048^3	–	1053.2	437.3

- 541 clauses (27 min/max)
- Very little CSG
- Many smooth blends (`exp`/`log`)

Based on a design by
Hazel Fraticelli and Anthony Taconi

AMORTIZED WORK (1024^3 VOXELS)



WHERE DO WE GO FROM HERE?

SPEED IS LIMITED BY RAM ACCESS

HPC

Feb 11, 2015

GPU Pro Tip: Fast Dynamic Indexing of Private Arrays in CUDA

By Maxim Milakov

Tags: **CUDA**, **CUDA C/C++**, **Pro Tip**

Discuss (8)

Using local memory is slower than keeping array elements directly in registers, but if you have sufficient math instructions in your kernel and enough threads to hide the latency, the local load/store instructions may be a minor cost. Empiricially, a 4:1 to 8:1 ratio of math to memory operations should be enough, the exact number depends on your particular kernel and GPU architecture. Your kernel should also have occupancy high enough to hide local memory access latencies.

CISC INSTRUCTIONS

- Improve the math-to-computation ratio by adding bigger opcodes
- For example:
  - Every 2D circle computes $sqrt((x - x_0)^2 + (y - y_0)^2) - r$
  - `RADIUS_IMM` → `slots[out] = sqrt(slots[lhs]² + slots[rhs]²) – imm`
  - Before:
    - 5 tape clauses
    - 5 slot reads
    - 5 slot writes
  - After:
    - 1 tape clause
    - 2 slot reads
    - 1 slot write

VARIABLE INSTRUCTION LENGTH

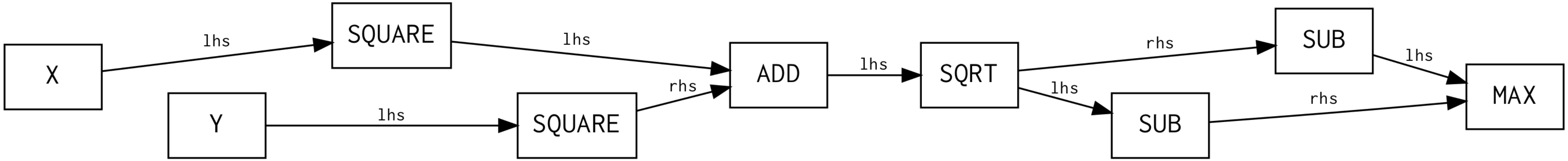
opcode	out	lhs	rhs	immediate / jump
u8	u8	u8	u8	f32 / i32

← 50% of clause size
Not always used!

- Using variable-length instructions would reduce memory traffic
- This is harder than it sounds!
- Tapes are decoded in both directions
  - Forward during evaluation
  - Backwards during shortening
- Need a strategy for unambiguous encoding

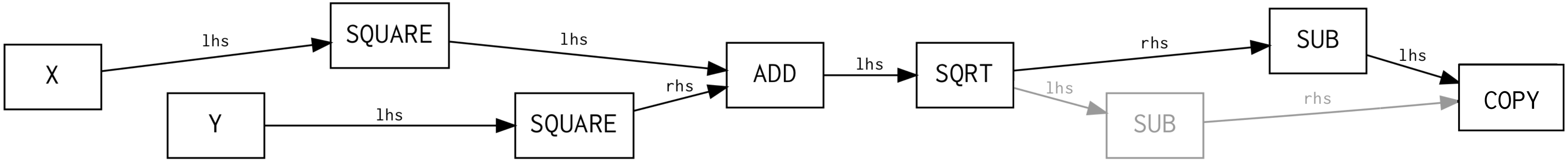
TAPE SHORTENING WITHOUT INTERPRETER

- Instead of building shortened tapes, compile a kernel with sections that are skipped based on flags



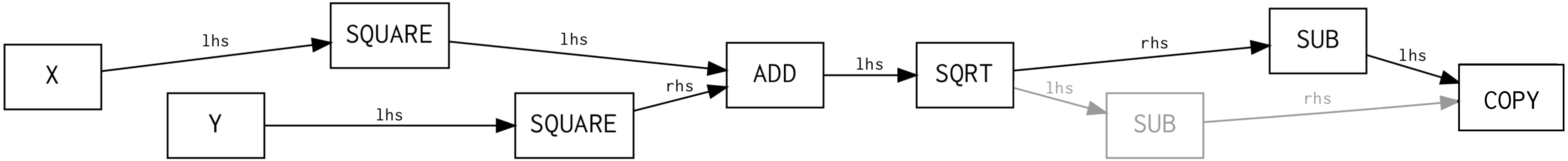
TAPE SHORTENING WITHOUT INTERPRETER

- Instead of building shortened tapes, compile a kernel with sections that are skipped based on flags



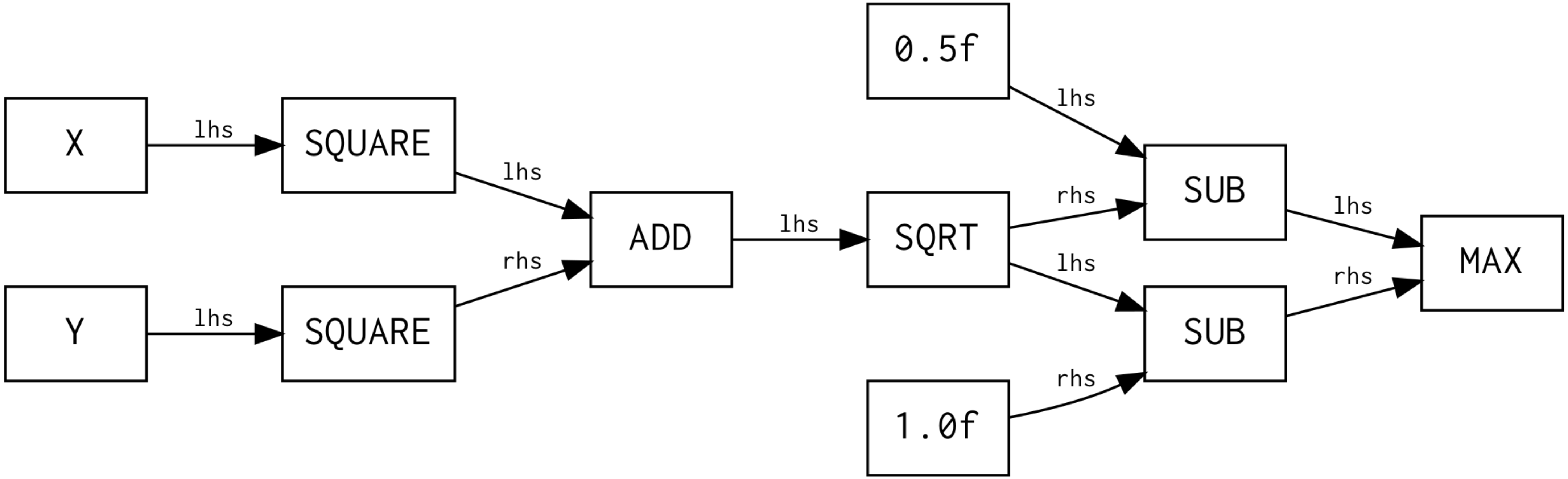
TAPE SHORTENING WITHOUT INTERPRETER

- Instead of building shortened tapes, compile a kernel with sections that are skipped based on flags

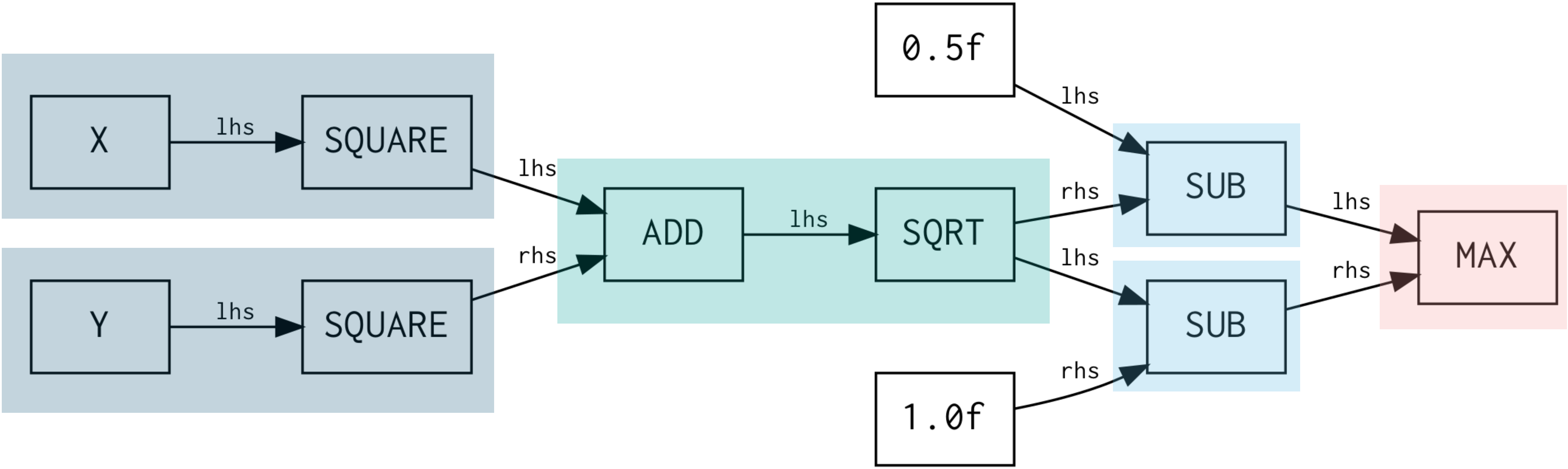


- How do you build the splitting points?
  – *O*(n^3) possibilities with min/max nodes
  – Interesting graph problem!
  – Draw from compiler literature?

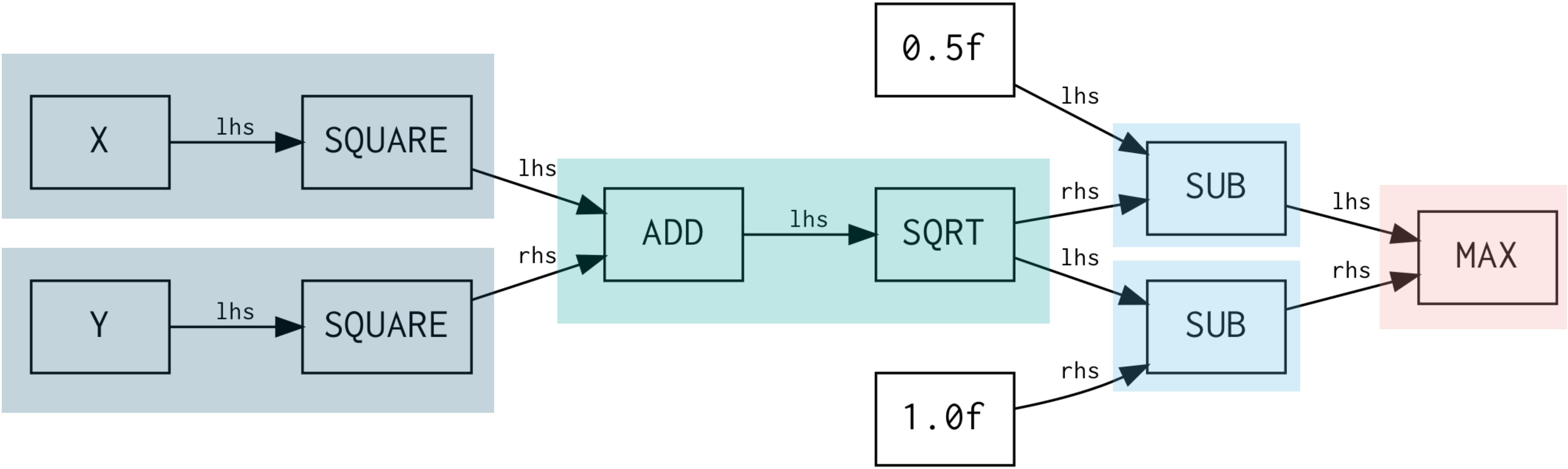
MORE PARALLELISM



MORE PARALLELISM



MORE PARALLELISM



- *Also* an interesting graph problem!
  - Postdominance frontiers?
  - Synchronization overhead?
  - Effectively scale to **even larger** GPUs

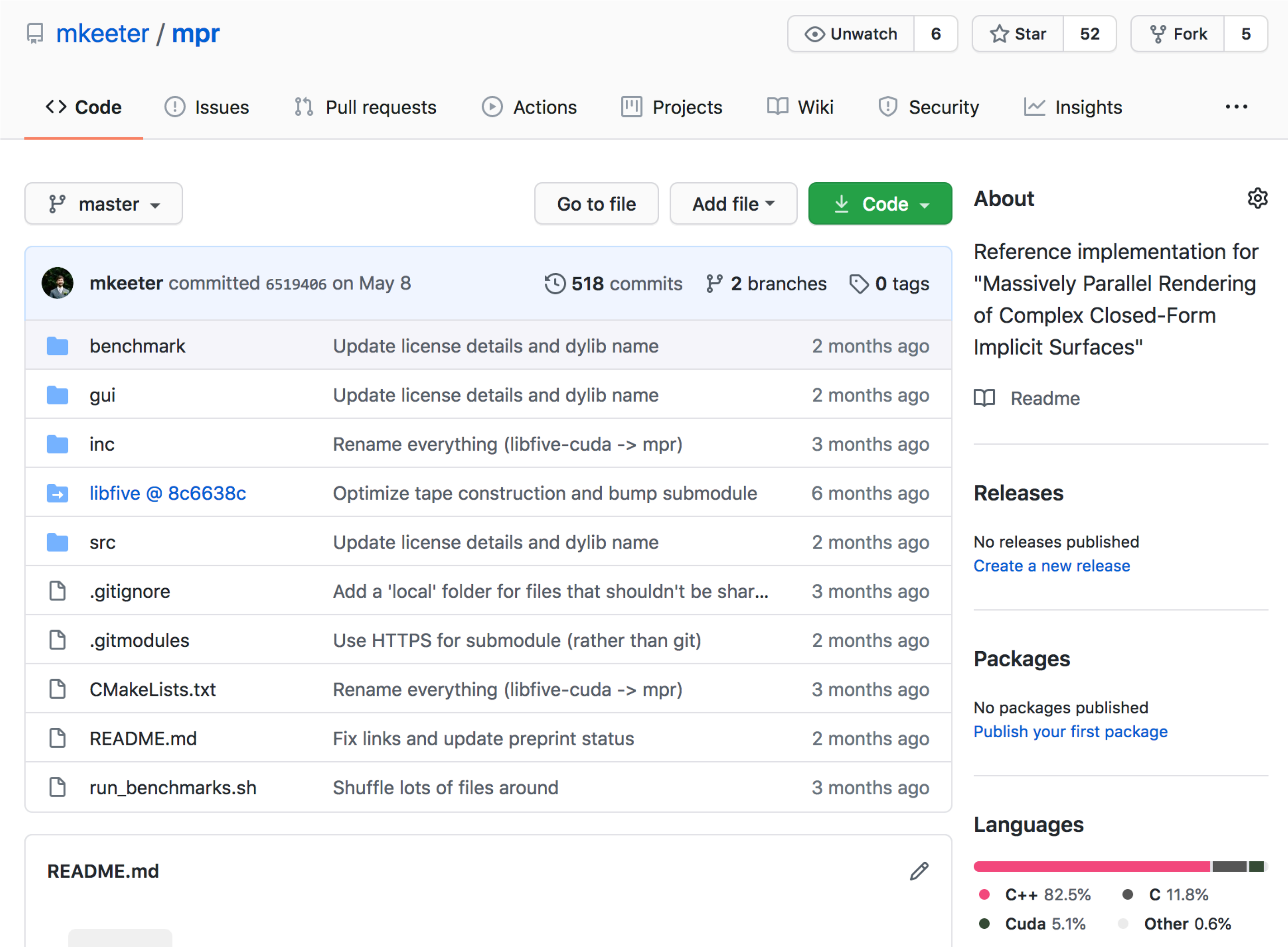
OTHER POTENTIAL FUTURE WORK

- Use this technique for fast voxelization, then render with other GPU-first algorithms for rendering voxel data
  - Sparse Voxel Octrees (Laine and Karras ’10)
  - GVDB (Hoetzlein ’16)
- Use reduced affine arithmetic for better intervals (Fryazinov et al. ’10)
- Better depth culling of tiles and voxels
- Implementing black-box “oracles” for interoperability with other model formats
  - `libfive` already does this for CPU-side evaluation

CONCLUSIONS

- New method for rendering complex closed-form implicit surfaces on the GPU
- No triangulation or conventional raytraycing!
- Limited to rendering pure mathematical shapes
- Scales well with GPU power
- Works best on hard-surface CSG
- Interpreter speed is gated by global RAM access
  - Being able to generate executable code on the GPU would be great!
- Lots of RAM required for storing shortened tapes

REFERENCE IMPLEMENTATION



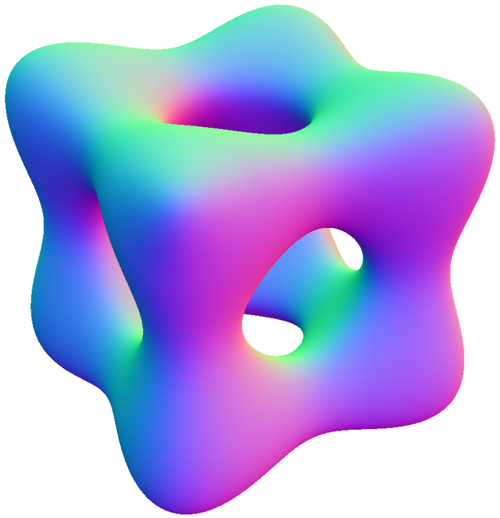
github.com/mkeeter/mpr

Released under MPL v2 (weak copyleft)

Reproduce my results on AWS for about $5!

ACKNOWLEDGEMENTS AND THANKS

- **Beta readers:** Jonathan Bachrach, Blake Courter, Martin Galese, Neil Gershenfeld, Raph Levien, Brian Merchant, Doug Moen, and Amira Abdel Rahman.
- **Benchmarking:** Martin Galese
- **Models:**
  - The bear head is based on a design by Hazel Fraticelli and Anthony Taconi
  - The involute curve math was derived by Peter Fedak
  - The architecture model is based on a design by Jennifer Keeter
- The anonymous reviewers for their feedback and insight
- My colleagues at **Formlabs**, for encouraging this work
- **nTopology**, for their support of the `libfive` kernel



THANK YOU!

Matt Keeter
Independent researcher

mattkeeter.com/research/mpr
matt.j.keeter@gmail.com
@impraxical