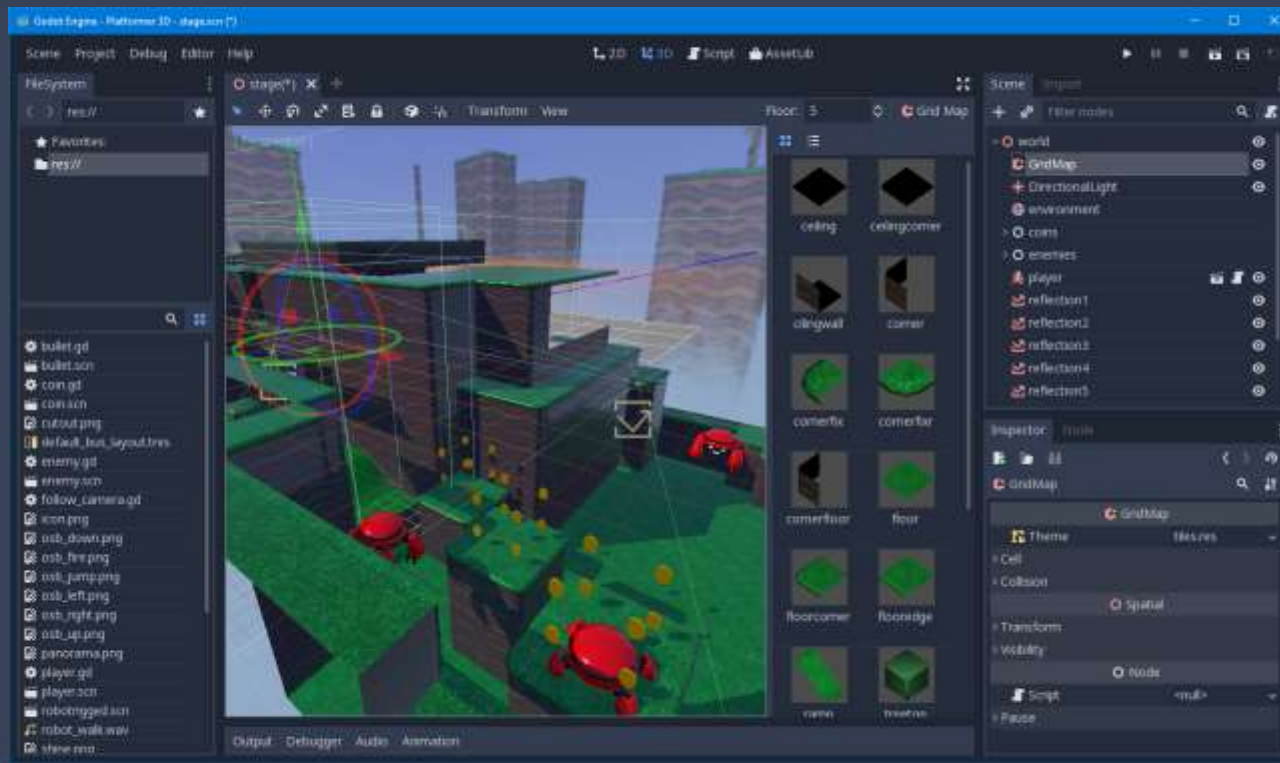
GLOBAL ILLUMINATION IN GODOT ENGINE

Pedro J. Estébanez
(a.k.a. RandomShaper)

CC BY-SA 4.0

FOUNDATION

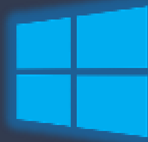
GAME ENGINE + EDITOR



- 2D & 3D
- Full of visual tools
- Very easy to learn

CROSSPLATFORM

- Desktop · · · · · ·
  (games + editor)



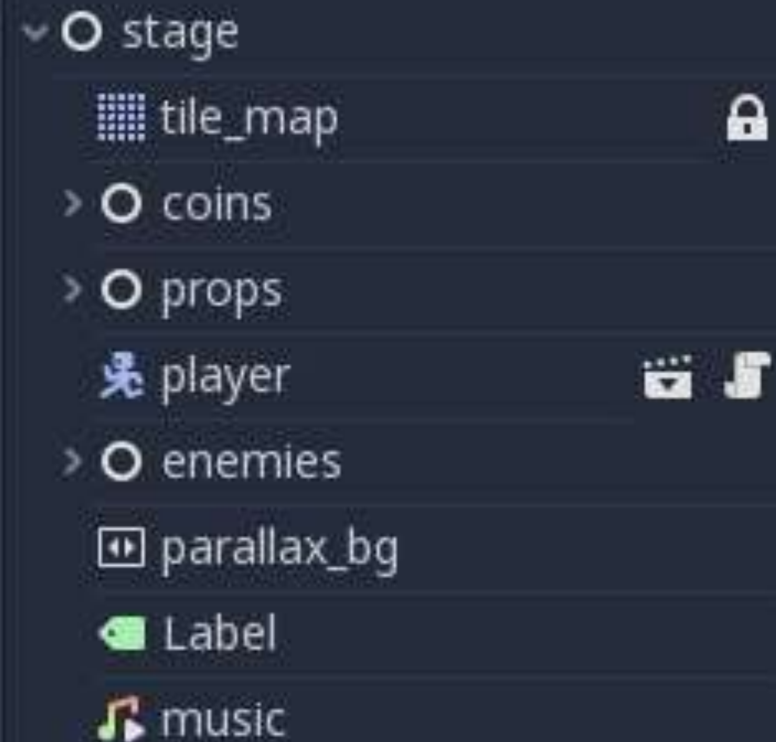
- Mobile · · · · · · ·

- Web · · · · · · · · ·
  (WebAssembly)

- Consoles · · · · · ·
  (via 3rd-parties)

EVERYTHING IS A NODE

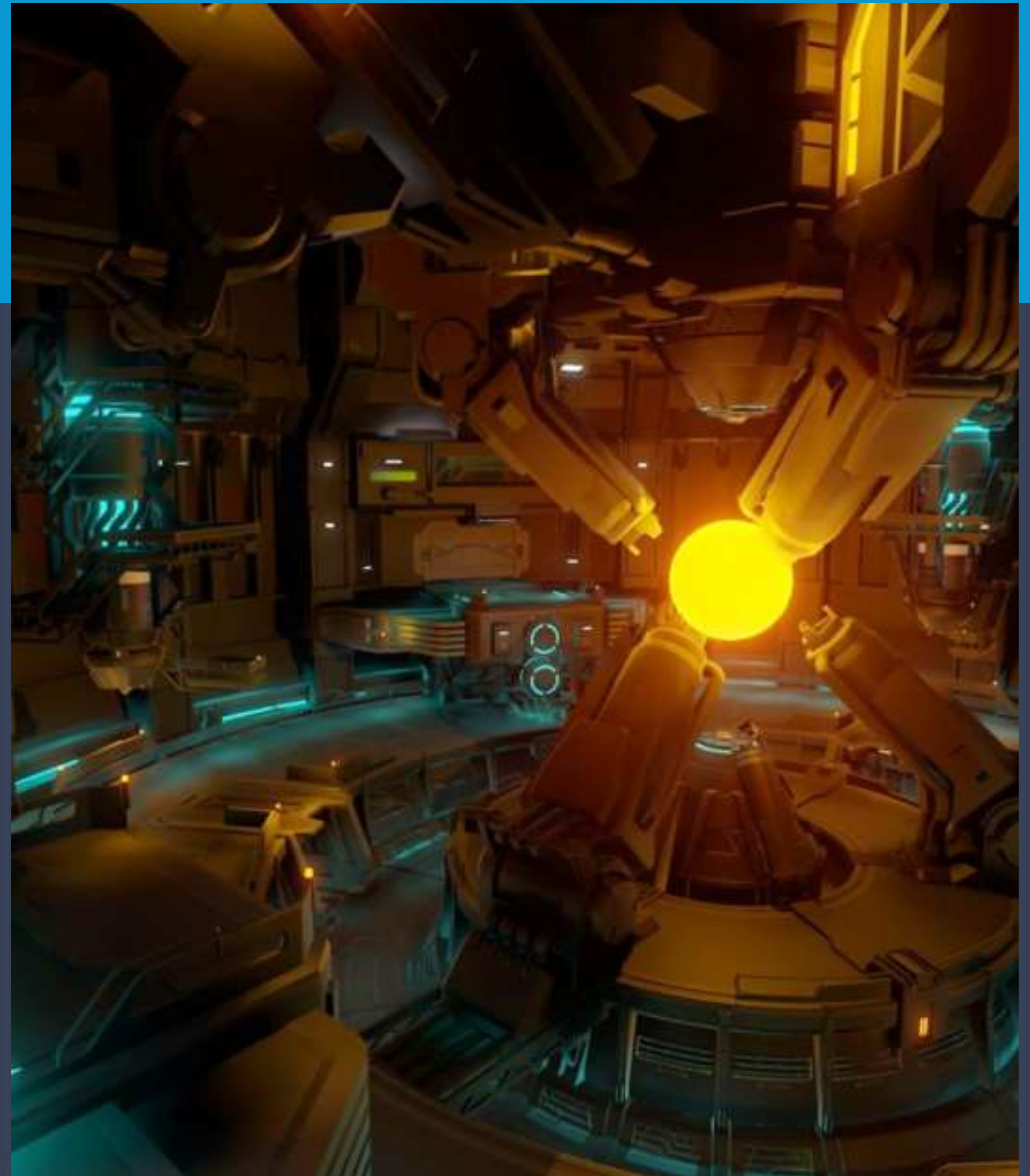
- A scene is a node, too
- Organization is promoted (instancing, inheriting, nesting)
- Low-risk parallel work for teams
- Programmers and artists can collaborate without interference



MAIN FEATURES

STATE-OF-THE-ART 3D

- Physics Based Rendering
- Real-time Global Illumination
- Most common effects supported (Tone-mapping, SSAO, SSR…)



VCS (AND HUMAN) FRIENDLY

```
[node name="detect_wall_right" type="Sprite"]

position = Vector2( 3.2788, -0.381488 )
rotation = -1.5708
enabled = true
exclude_parent = true
cast_to = Vector2( 0, 20 )
collision_mask = 1
type_mask = 15

[node name="detect_floor_right" type="RayCast2D" parent="

position = Vector2( 29.1987, -9.34363 )
enabled = true
exclude_parent = true
cast_to = Vector2( 0, 45 )
collision_mask = 1
type_mask = 15
```

- Almost everything consists in plain text files
- VCS-friendly format (nice diffs)
- Easy to read/edit by humans alike

MULTIPLE LANGUAGES

- GDScript
- C# 7.0 (via Mono)
- Visual Scripting
- C++ (NativeScript)
- Unofficial: Nim, Rust, Python, D

```
19
20 #cache the sprite here for fast access (we will set sc
21 onready var sprite = $sprite
22
23 func _physics_process(delta):
24     #increment counters
25
26     onair_time += delta
27     shoot_time += delta
28
29     ### MOVEMENT ###
30
31     # Apply Gravity
32     linear_vel += delta * GRAVITY_VEC
33     # Move and Slide
34     linear_vel = move_and_slide(linear_vel, FLOOR_NORM
35     # Detect Floor
36     if is_on_floor():
37         onair_time = 0
38
39     on_floor = onair_time < MIN_ONAIR_TIME
40
41     ### CONTROL ###
42
43     # Horizontal Movement
44     var target_speed = 0
45     if Input.is_action_pressed("move_left"):
```

EXTENSIBLE

- Plugin architecture
- Asset library
- GDNative

DEVELOPMENT

OPEN SOURCE

- Open-sourced in 2014
- Hosted on GitHub
- Travis + AppVeyor CI
- MIT License

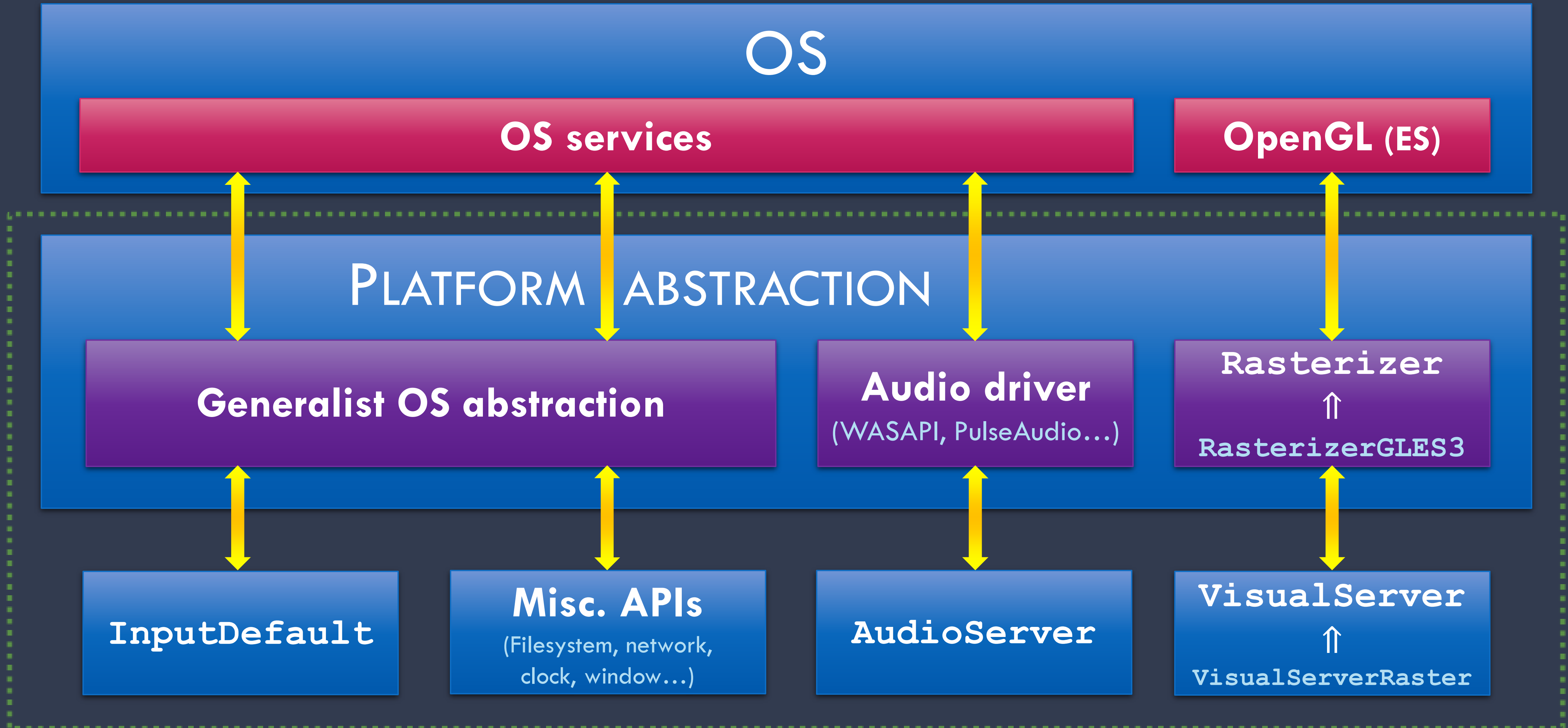
THE CODE ITSELF

```
void Node::_notification(int p_notification) {

	switch (p_notification) {

		case NOTIFICATION_PROCESS: {

			if (get_script_instance()) {

				Variant time = get_process_delta_time();
				const Variant *ptr[1] = { &time };
				get_script_instance()->call_multilevel(SceneStringNames::get_singleton()-
			}
		} break;
		case NOTIFICATION_PHYSICS_PROCESS: {

			if (get_script_instance()) {

				Variant time = get_physics_process_delta_time();
				const Variant *ptr[1] = { &time };
				get_script_instance()->call_multilevel(SceneStringNames::get_singleton()-
			}

		} break;
		case NOTIFICATION_ENTER_TREE: {

			if (data.pause_mode == PAUSE_MODE_INHERIT) {

				if (data.parent)
					data.pause_owner = data.parent->data.pause_owner;
				else
					data.pause_owner = NULL;
			} else {
				data.pause_owner = this;
			}

			if (data.input)
				add_to_group("_vp_input" + itos(get_viewport()->get_instance_id()));
			if (data.unhandled_input)
				add_to_group("_vp_unhandled_input" + itos(get_viewport()->get_instance_id
			if (data.unhandled_key_input)
				add_to_group("_vp_unhandled_key_input" + itos(get_viewport()->get_instanc
```

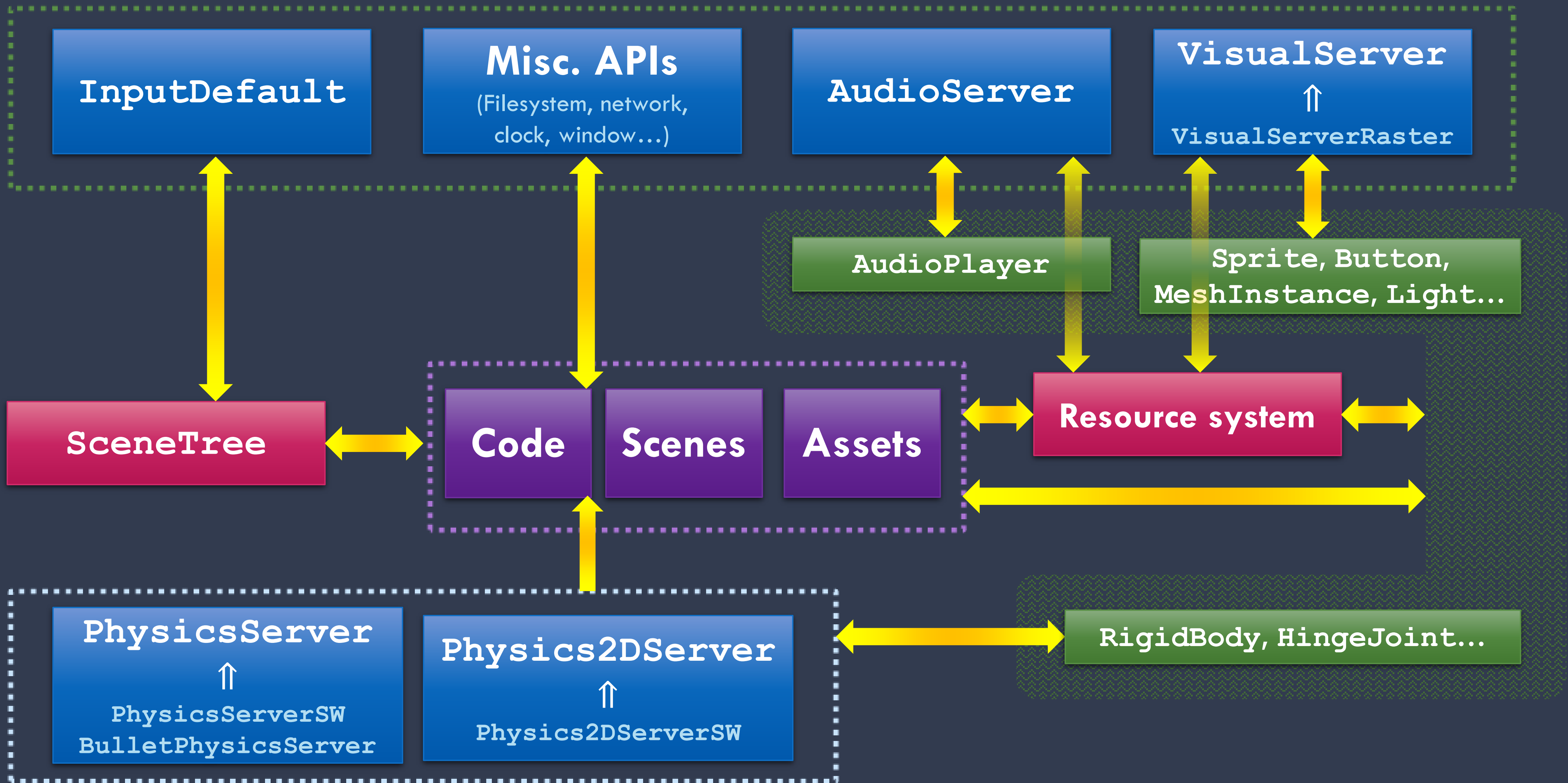
- SCons build system
- ~1 M lines of C++03
- `clang-format`ted
- Not using STL
- Custom containers and `String` (`Vector`, `List`, `Map`...)
- Very dev-friendly

ARCHITECTURE

ENGINE ⟷ OUTSIDE WORLD



ENGINE ⟷ GAME

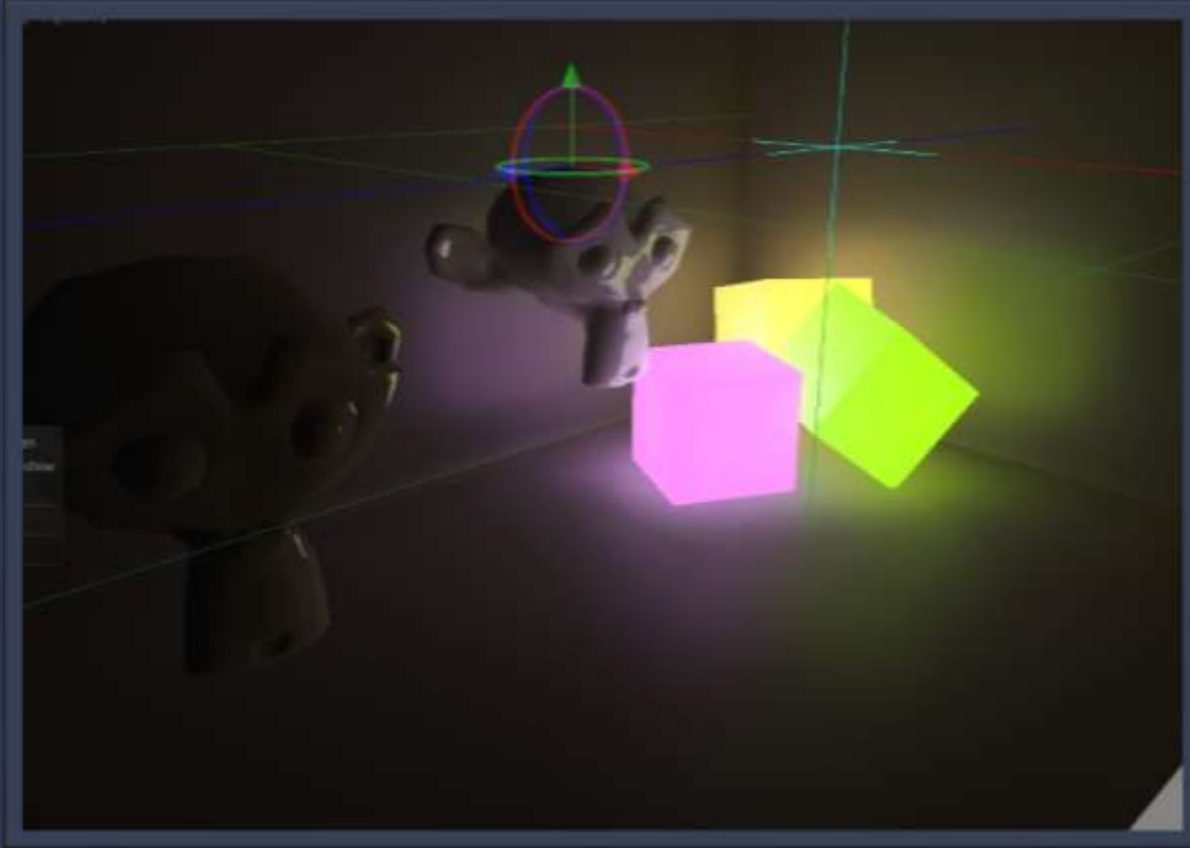


GLOBAL ILLUMINATION (GI)

GI CONTRIBUTION

Reflection probe + SSAO + Glow
+ **GI probe**

GI CONTRIBUTION



Emissive materials

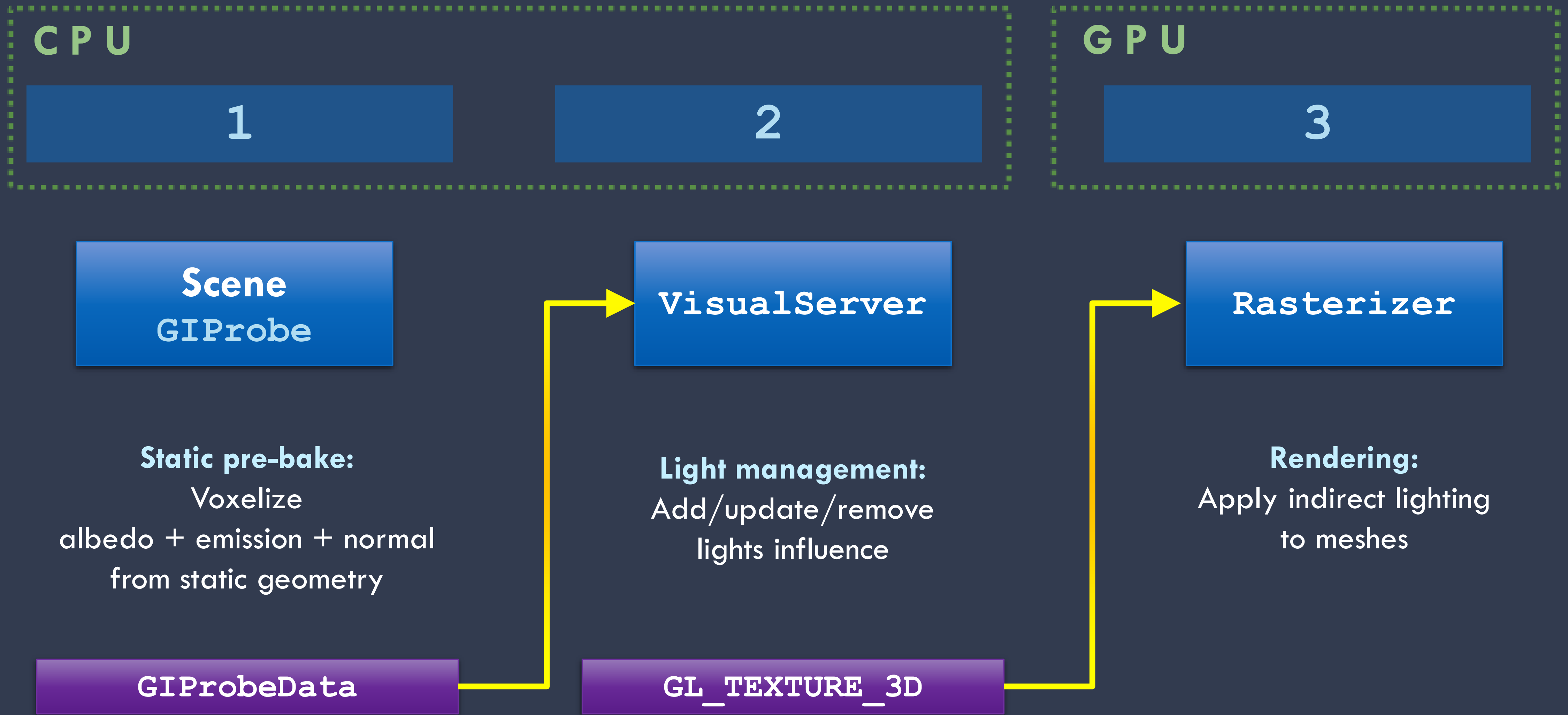
Reflections

TRADEOFFS

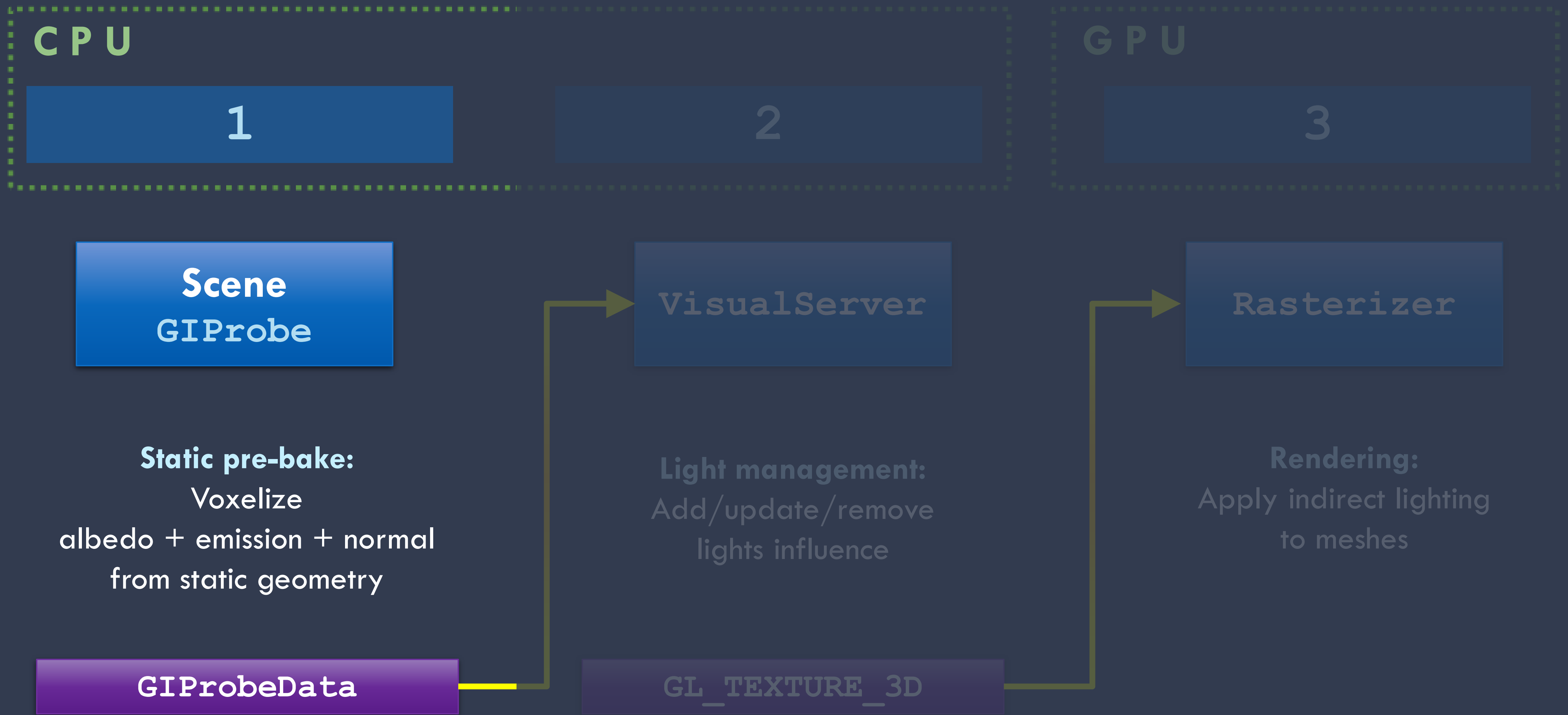
- High quality indirect light
- Not-so-costly to render
- Dynamic objects are affected in real time

- Memory-hungry
- Too expensive for mobile
- Voxelization can be evident

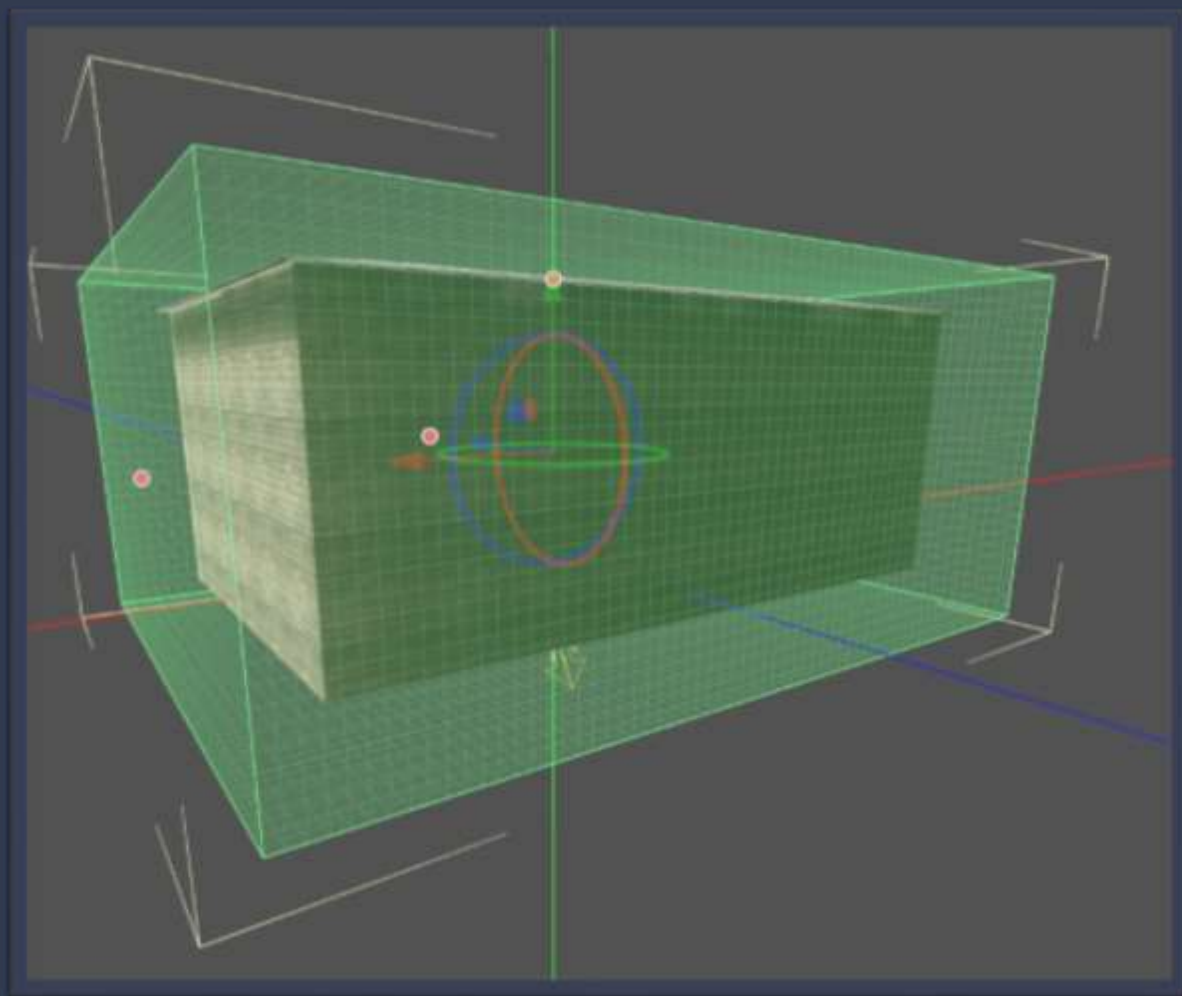
PROCESS



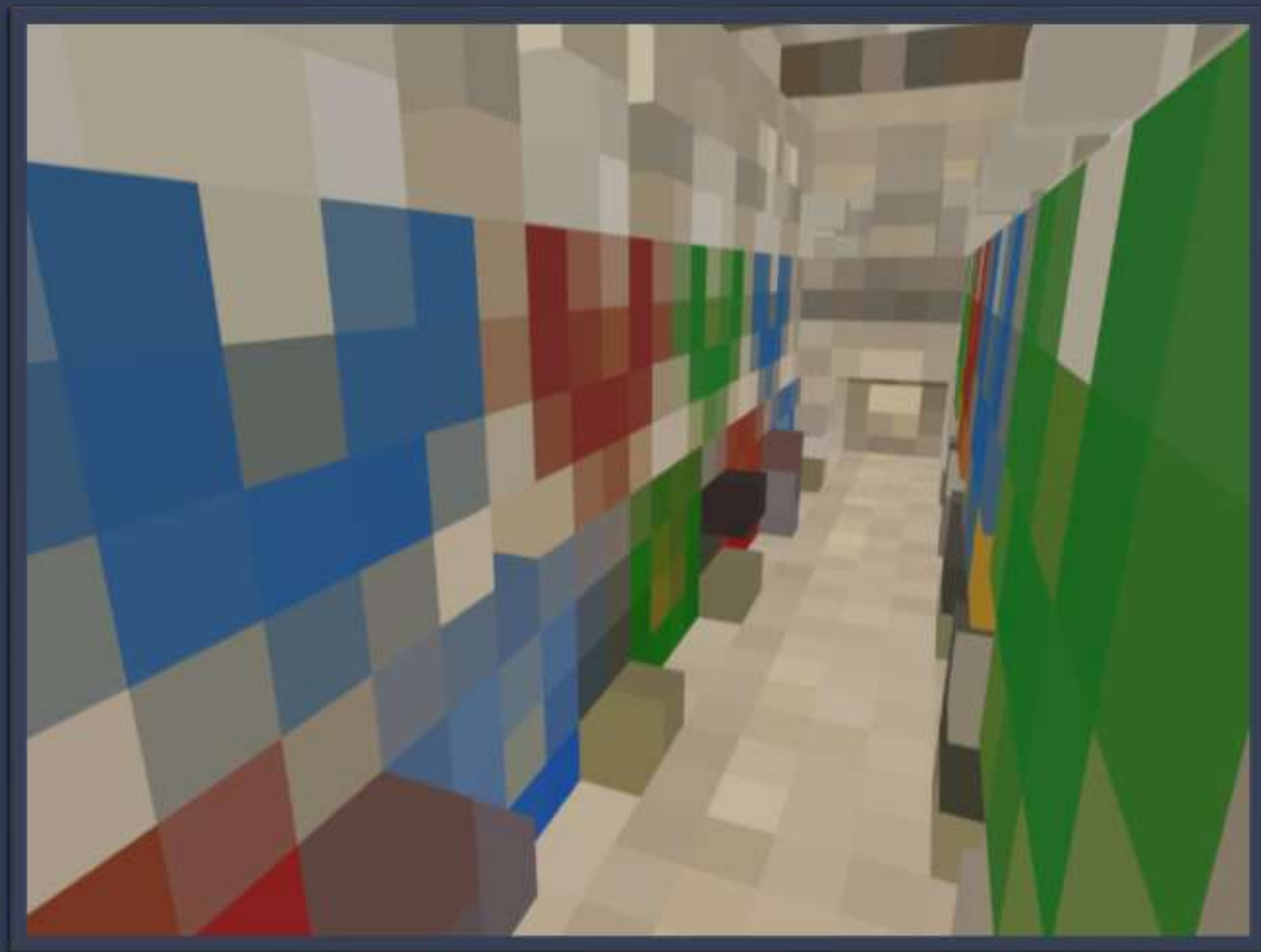
1. GIProbe



1. GIProbe



1. GIProbe



- Voxels always cubic
- 2^n voxels on every axis

1. `GIProbe`

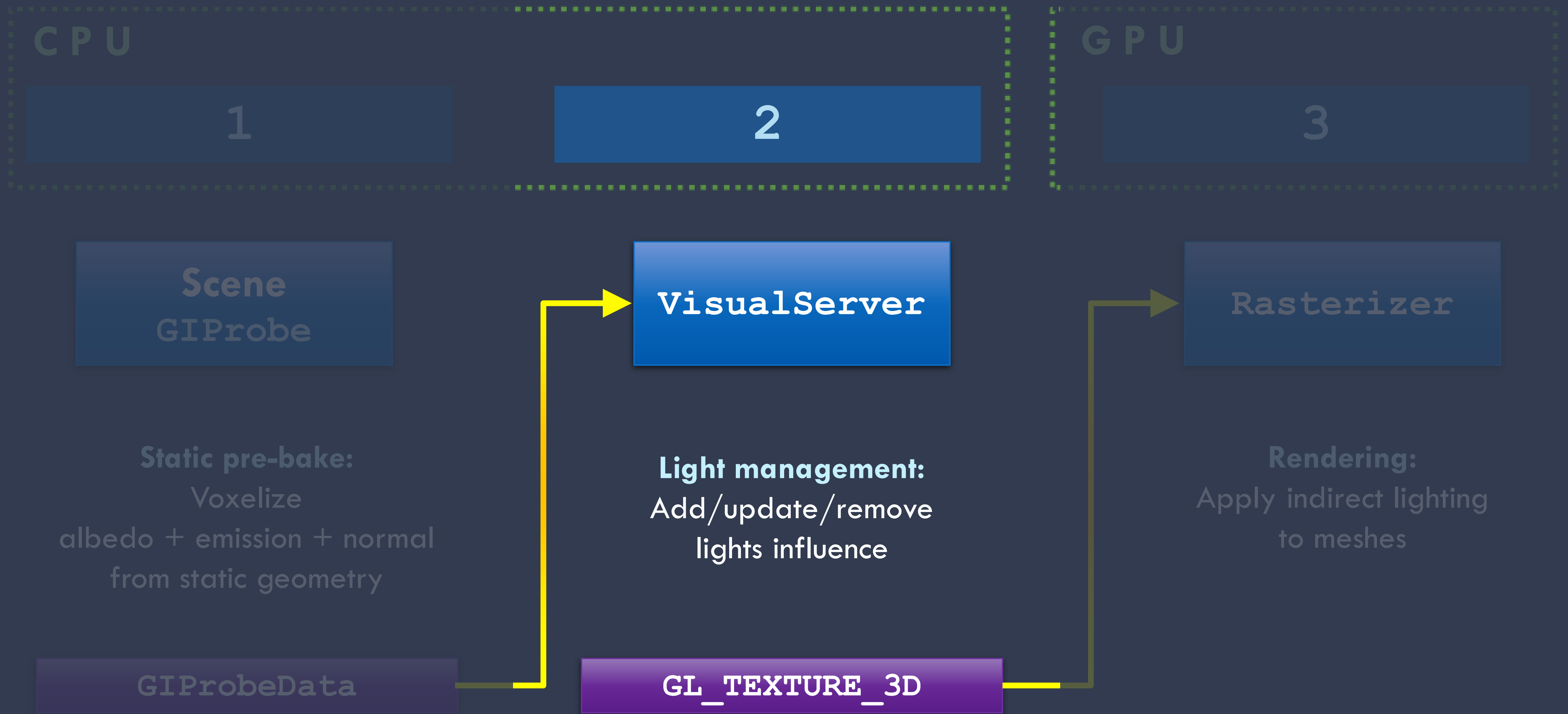
Pre-bake (usually done at the editor):

- Per every affected `MeshInstance` (intersecting AABB)
  - per surface, per face
    - `plot(face, root_cell, 0, probe_aabb)`
- `fixup() // Average alphas (presence coefficient)`
- Serialize to a BLOB (`GIProbeData`, `PoolVector<int>`)
  - clamp to [0, 1]
  - `float` to `uint8_t`

1. GIProbe

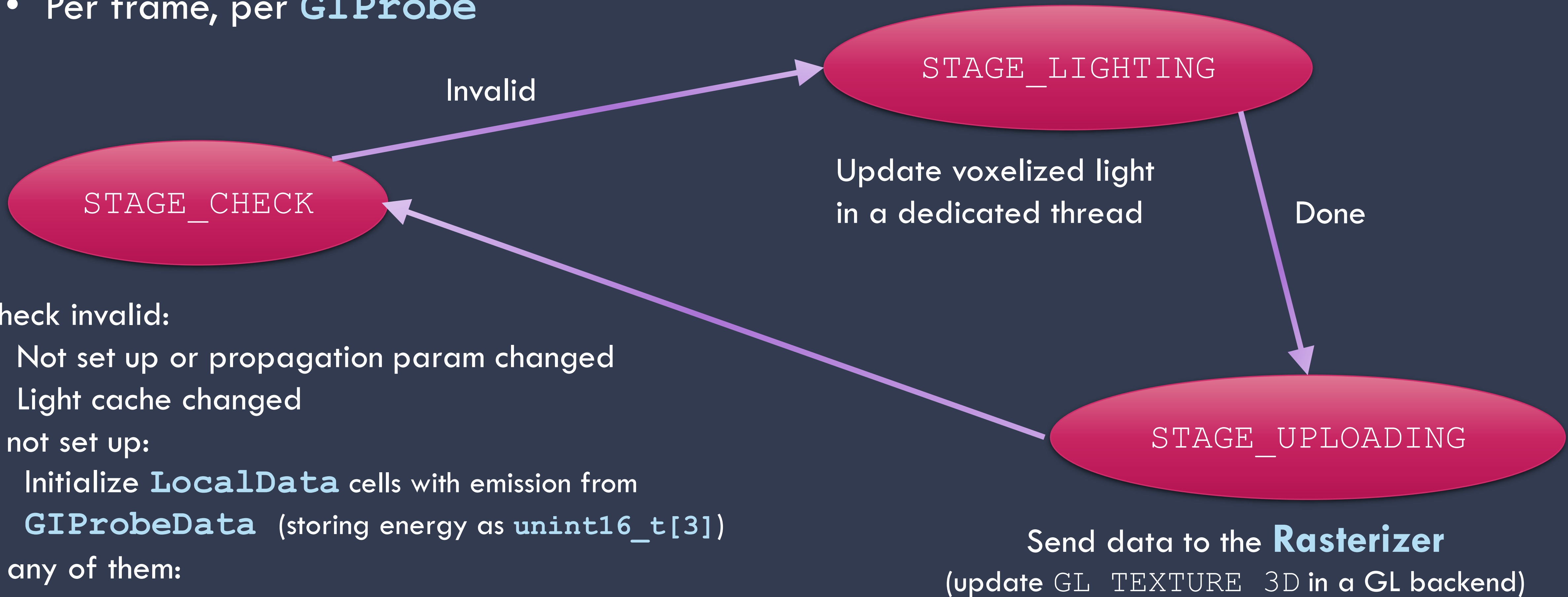
```
plot(face, cell, level, aabb):
    if level < max_level:
        for child_cell of split(cell):
            if not face in aabb skip
            plot(face, child_cell, level + 1, child_cell.aabb)
    else:
        subcell_data = 0 // albedo, emission, normal, alpha
        for subcell of subdivide(cell, 4, 4, best_axis(face)):
            if not face in subcell.aabb skip
            subcell.aen += raycast_sample(…)
            subcell.alpha += 1
        cell.data += subcell_data / (4 * 4)
```

2. VisualServer



2. VisualServer

- Per frame, per GIProbe



2. `VisualServer`

`STATE_LIGHTING`:

Update voxelization in `local_data` upon scene `GIProbeData` and lights:

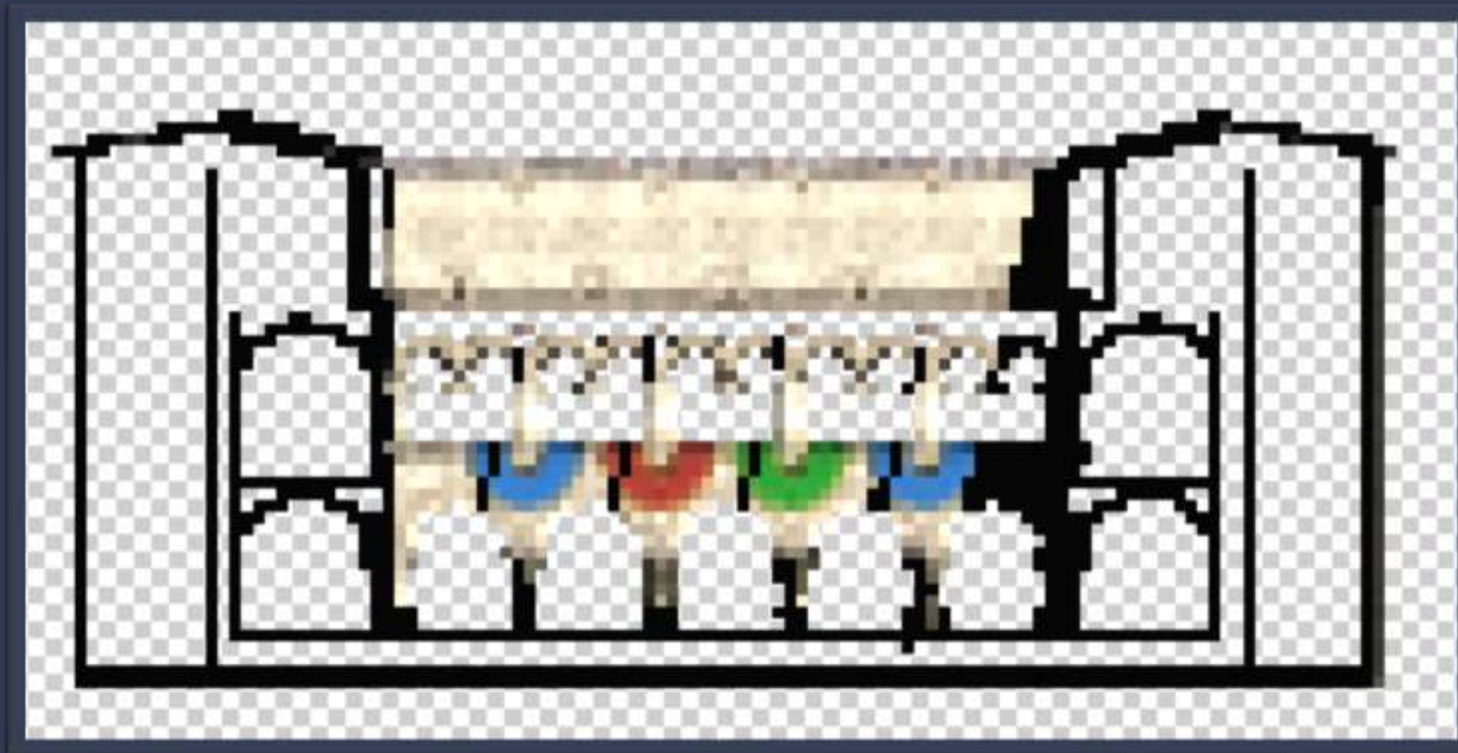
- Per each light removed or changed
  ```
  bake_light(light, -1) // Deterministic
  ```
- Per each ight added or changed
  ```
  bake_light(light, +1) // Deterministic
  ```
- Recursively downsample up to the first level
- Build final data, ready to be sent to the `Rasterizer`
  (even right as S3TC if wanted):
  - Mip-map chain, each texel computed as:
    ```
    texel.rgb = local_cell.energy / probe.dyn_range
    texel.a = probe_cell.alpha
    ```

2. VisualServer

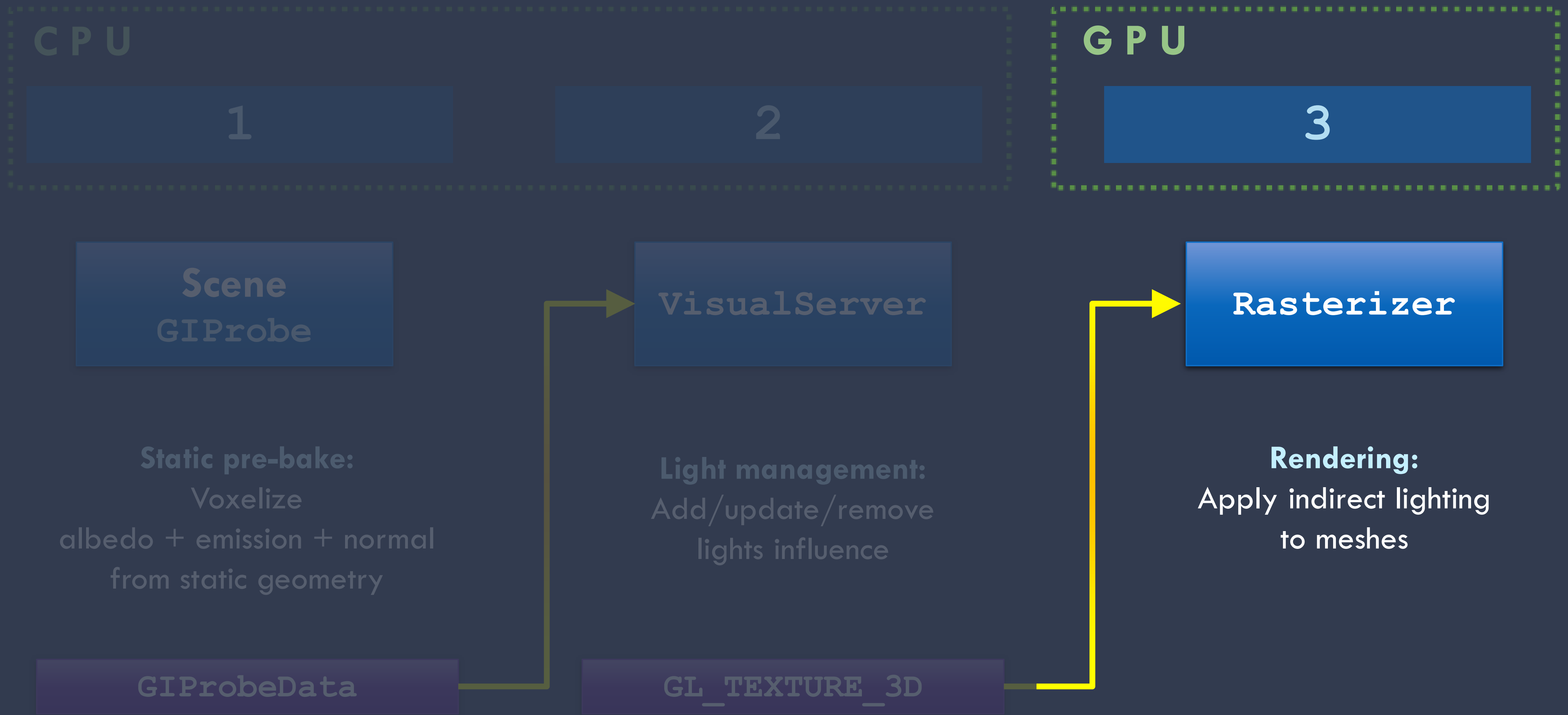
```
bake_light(light, sign):
    for each cell:
        cell_hit = track(light, cell)
        if cell_hit == cell:
            light_rgb = light.color * light.energy
            local_data_cell.energy +=
                gi_probe_data.cell.albedo * light_rgb
```

2. VisualServer

Once slice of the GL_TEXTURE_3D (128x64x128)



3. Rasterizer



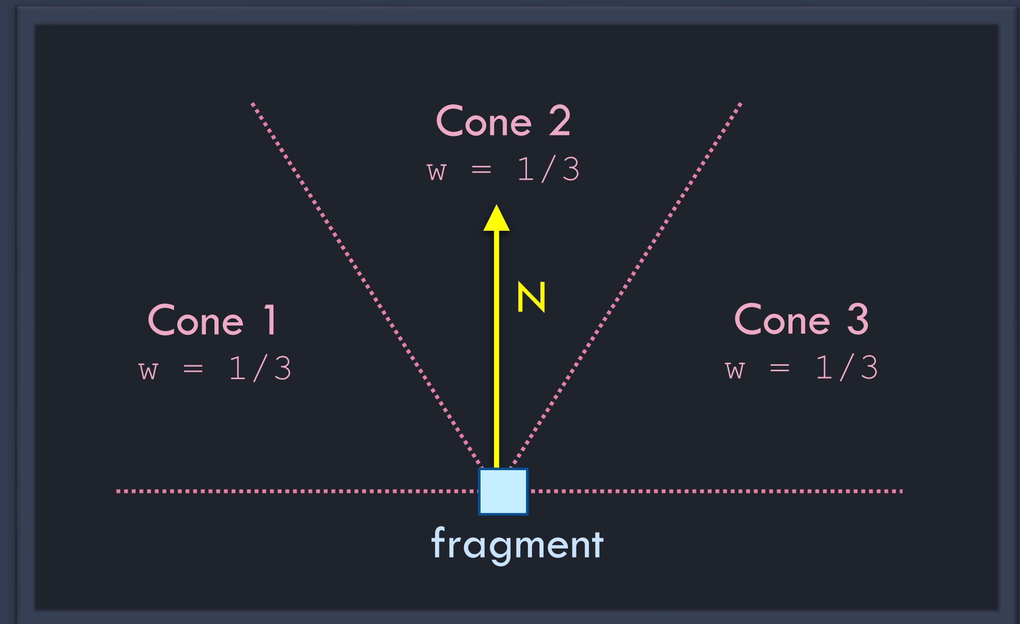
3. `Rasterizer`

- Per frame, `VisualServer` associates `GIProbes` with affected meshes.
- The `Rasterizer` picks two at most for a given mesh, and sets these uniforms:

```
uniform mediump sampler3D gi_probe1;
uniform highp mat4 gi_probe_xform1;
uniform highp vec3 gi_probe_bounds1;
uniform highp vec3 gi_probe_cell_size1;
uniform highp float gi_probe_multiplier1;
uniform highp float gi_probe_bias1;
uniform highp float gi_probe_normal_bias1;
uniform bool gi_probe_blend_ambient1;

# The same for the 2nd probe (gi_probe...2)
```

3. Rasterizer

- Per every GI-affected mesh:
  - Per every fragment:
    - Sum indirect lighting, computed as a weighted average of the light inside N cones
- **Low quality:** N = 4 cones
- **High quality:** N = 6 cones
- In any case, 1 more for specular



3. Rasterizer

Voxel Cone Tracing

At each step:

- Accumulate alpha
- Accumulate `textureLod(pos, LOD) * (1.0 - accum_alpha)`

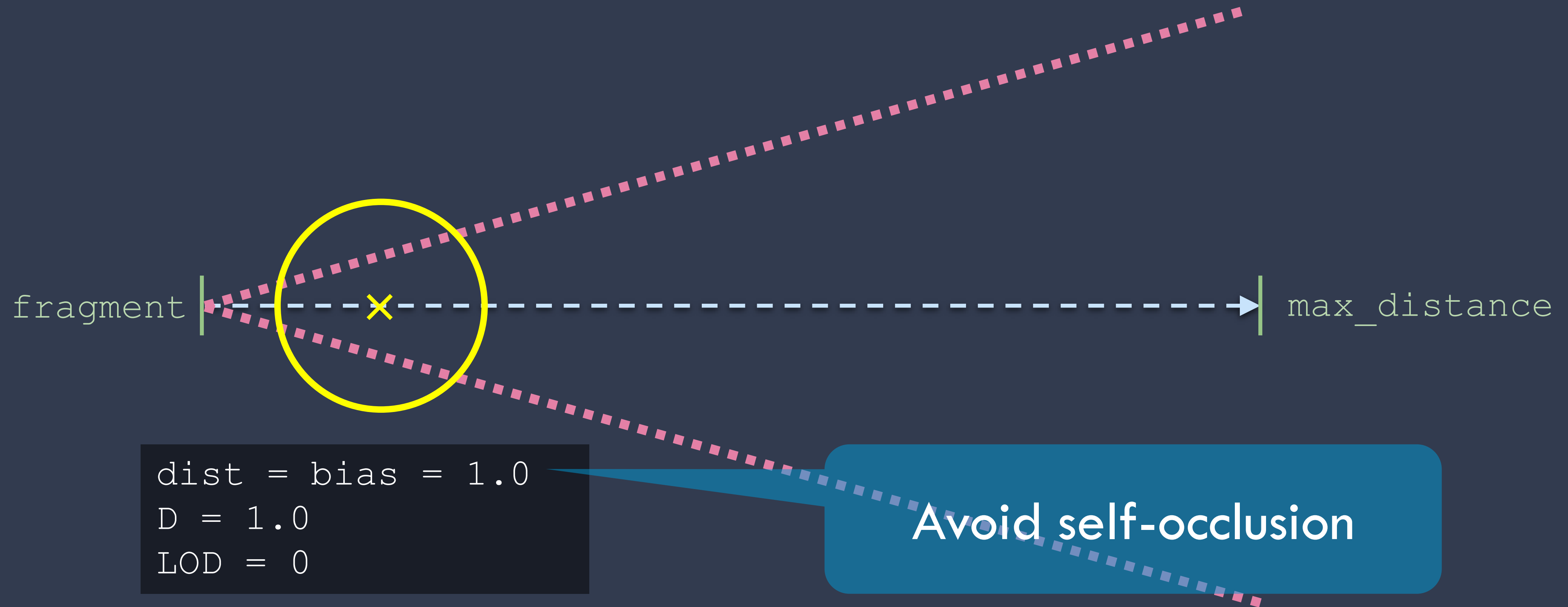
fragment → max_distance

Unit is voxel side

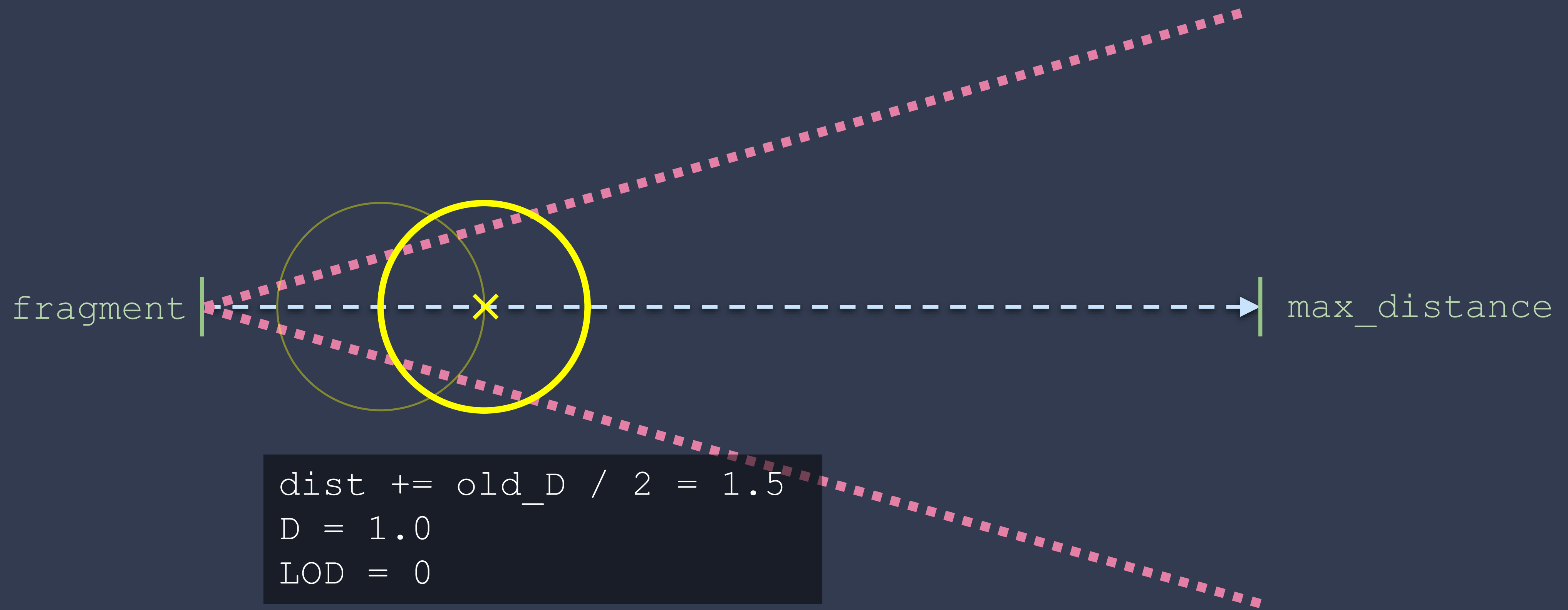
Exit condition:
`dist > max_distance || accum_alpha >= 0.95`

```
D = diameter = max(1.0, 2 * tan(angle / 2) * dist)
LOD = log2(D)
```

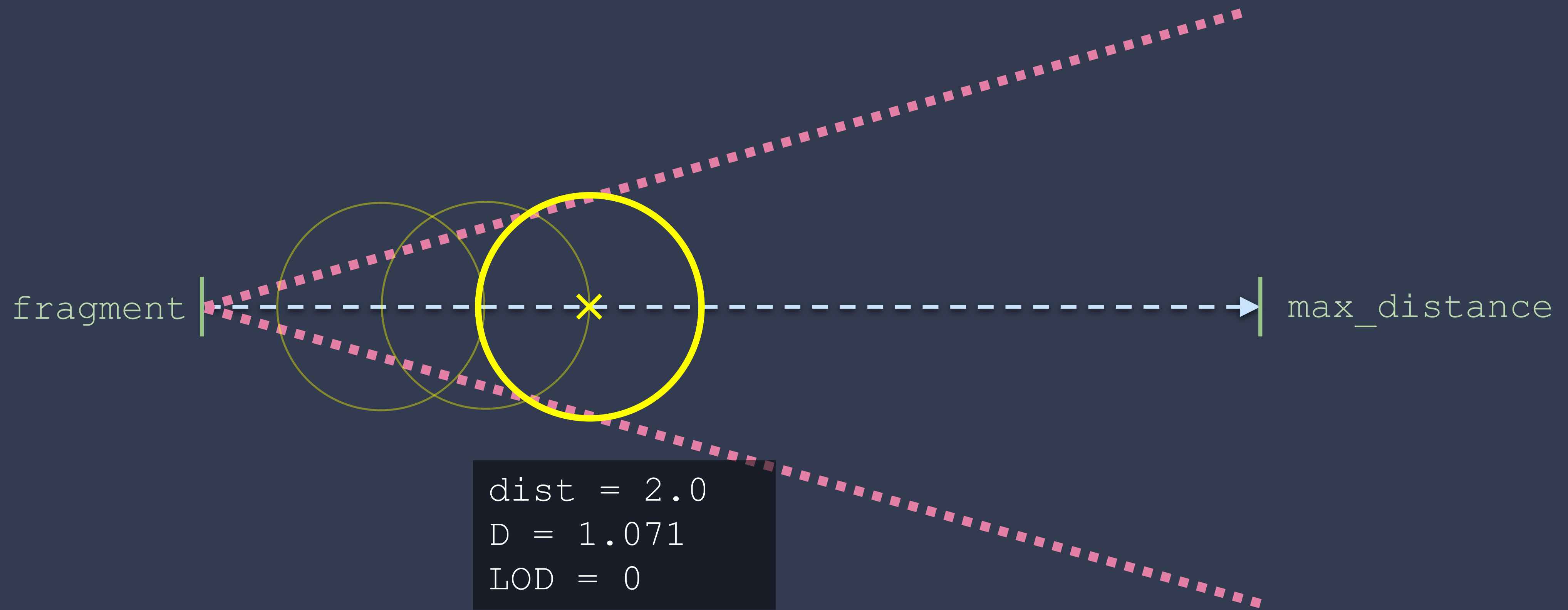
3. Rasterizer



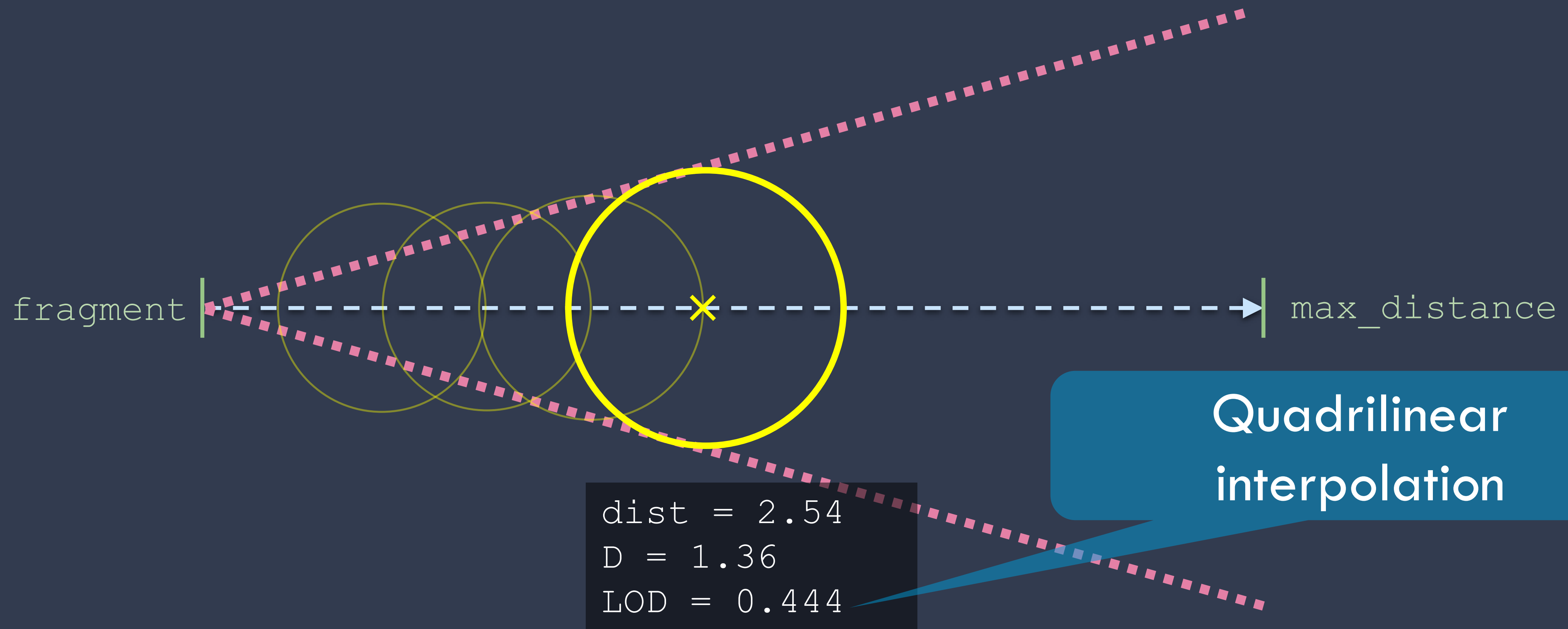
3. Rasterizer



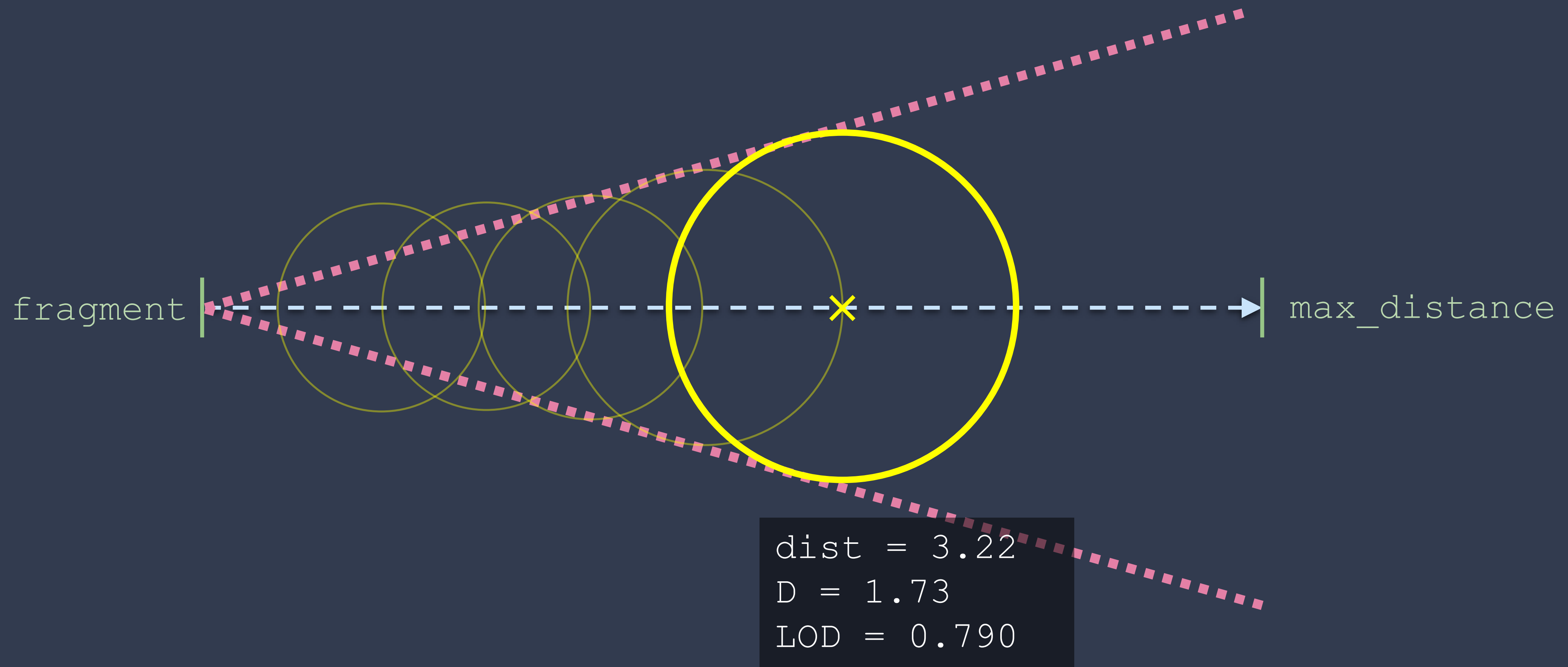
3. Rasterizer



3. Rasterizer



3. Rasterizer



THANK YOU!

QUESTIONS?

THANK YOU!

See you soon at

https://randomshaper.blogspot.com

Support Godot on

PATREON