LUND UNIVERSITY | LTH FACULTY OF ENGINEERING

PBRT 2:
Creating and Working with Large Scenes

Gustaf Waldemarson



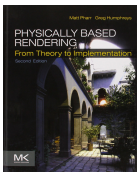
BCON26

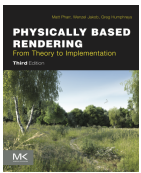
Who am I?

- (Former) Industrial PhD Student at the Lund University Graphics Group
- Compiler Performance Engineer at Arm











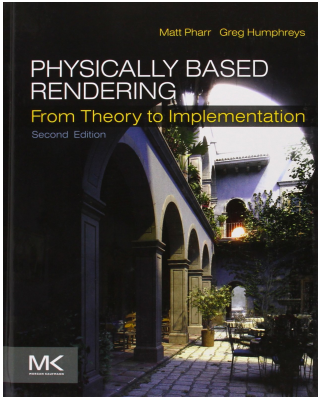
https://gustafwaldemarson.com

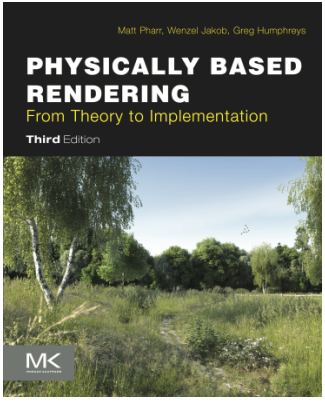
Agenda

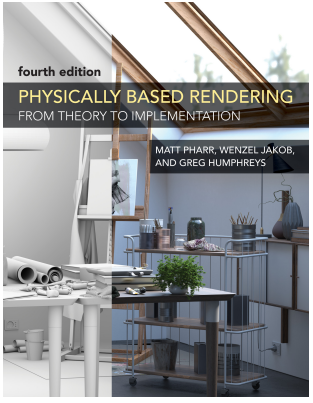
1. What is PBRT?
   - BCON24
   - Renders: PBRT vs Cycles
2. Creating Scenes
3. Large Scenes and Interactivity
4. Curves, Materials, and Volumes
5. Limitations



What is PBRT?

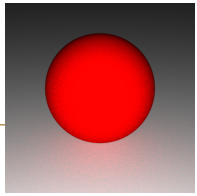






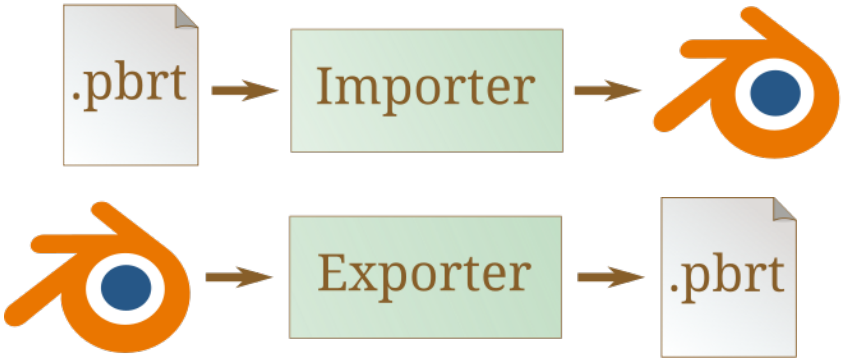
https://pbr-book.org/3ed-2018/contents
https://pbr-book.org/4ed/contents

The PBRT File Format

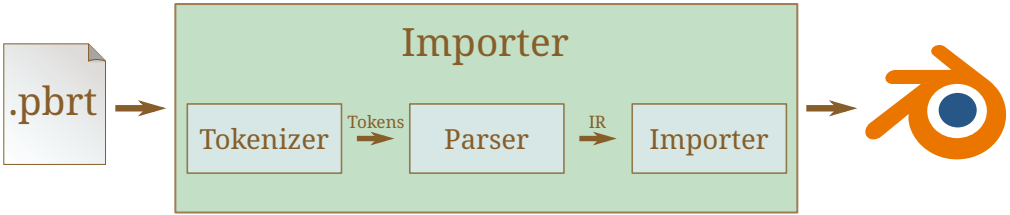


```
LookAt 0 5 8  0 .8 0   0 1 0
Film "image"
     "integer xresolution" [400] "integer yresolution" [400]
     "string filename" "sphere-ex.exr"
Camera "perspective" "float fov" [22]
Integrator "path"
WorldBegin
    AttributeBegin
        CoordSysTransform "camera"
        LightSource "point" "color I" [300 300 300]
    AttributeEnd
    Include "mesh.pbrt"
    Include "sphere.pbrt.gz"
WorldEnd
```

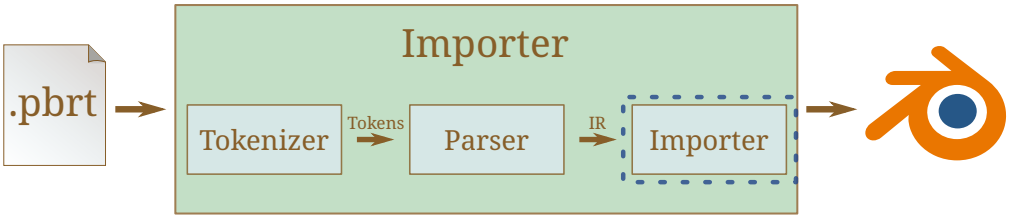
Recapitulating



Recapitulating



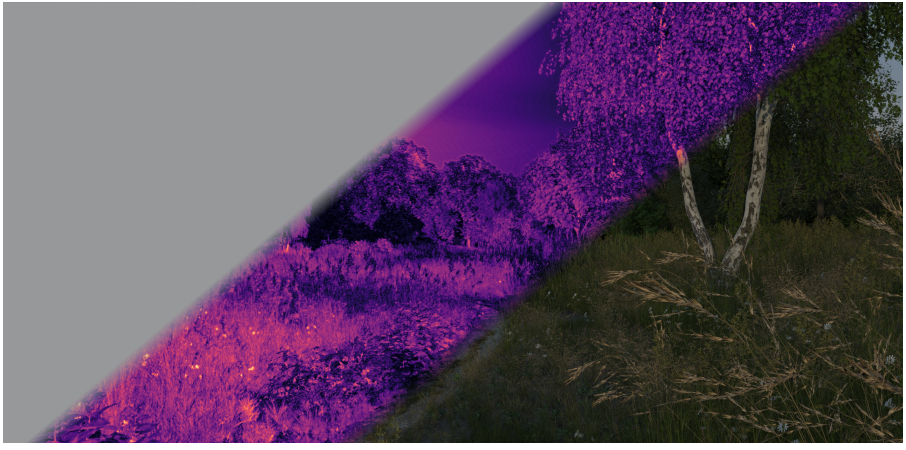
What's New?



Some Results — I



Some Results — II



Starting Off — The Intermediate Representation

```
world = {
    "lights": [...],
    "shapes": [...],
    "textures": [...],
    "materials": [...],
    "named_materials": {...},
    "named_objects": {...},
    "instances": [...],
}
```

Creating the Scene — Overview

```
material_map = {}
for index, material in enumerate(world["materials"]):
    material_map[index] = material.factory(material)
for s in world["shapes"]:
    shape.factory(s, material_map)
for al in world["area_lights"]:
    area_lights.factory(al)
for ls in world["lights"]:
    lights.factory(ls)

set_cycles_render_settings(pbrt_settings, world)
```

Creating Shapes — I

PBRT

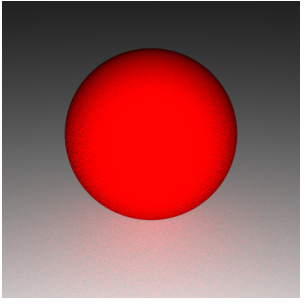
```
Translate 0 1 0
Rotate 60 1 1 1
Material "uber" "color Kd" [.8 .0 .0]
Shape "sphere" "float radius" [5.0]
```

Python

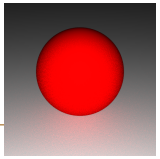
```
class Shape(NamedTuple):
    transform: list[list[float]]
    material: int
    type: str
    parameters: dict
```

Creating Shapes — II

```
def shape(s, import_settings):
    SHAPE_BUILDER = {
        "sphere": sphere,
        "trianglemesh": trianglemesh,
        "plymesh": plymesh,
        # ...
    }
    build = SHAPE_BUILDER[s.type]
    return build(s, import_settings)
```



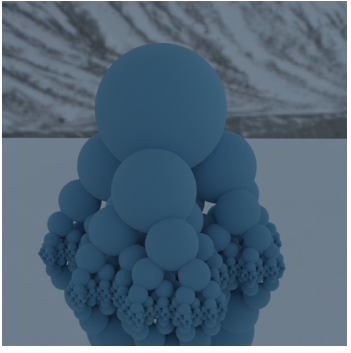
Creating Shapes — III — Sphere



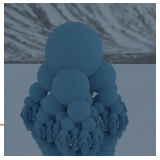
```
def sphere(s, import_settings):
    """Convert a PBRT sphere to a Blender shape."""
    radius = s.parameters.get("radius", [1.0])[0]
    kwargs = dict(radius=radius)
    bl_op = bpy.ops.mesh.primitive_uv_sphere_add(**args)
    return obj
```

Blender Operators Can Be Slow!

```
AttributeBegin
    Translate 0.5 0.5 0.5
    Shape "sphere" "float radius" [1]
AttributeEnd
AttributeBegin
    Translate 1.36603 -0.5 0
    Shape "sphere" "float radius" [0.6]
AttributeEnd
# 100 000 more spheres...
```

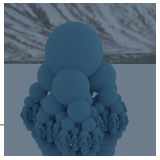


Caching — I



```
def cached_creation(name, operation, arguments, cache):
    """Create a new object or copy from the cache."""
    key = tuple(arguments.items())
    if key in cache:
        me_copy = cache[key].data.copy()
        return bpy.data.objects.new(name, me_copy)
    operation(**arguments)
    obj = bpy.context.active_object
    obj.name = name
    cache[key] = obj
    return obj
```

Caching — II



```
def sphere(s, cache, import_settings):
    """Convert a PBRT sphere to a Blender shape."""
    radius = s.parameters.get("radius", [1.0])[0]
    args = dict(radius=radius)
    bl_op = bpy.ops.mesh.primitive_uv_sphere_add
    obj = cached_creation("Sphere", bl_op, args, cache)
    return obj
```

Instances — I



Instances — II



PBRT

```
ObjectBegin "Mesh.000"
    Shape "sphere"
ObjectEnd

Transform [...]
ObjectInstance "Mesh.000"
Transform [...]
ObjectInstance "Mesh.000"
```

Python

```
class Instance(NamedTuple):
    transform: list[list[float]]
    name: str

world = {
    "named_objects": {...},
    "instances": [...],
}
```

Instances — III



```
collection_map = {}
for group_name, shapes in world["named_objects"].items():
    group = bpy.data.collections.new(group_name)
    collection_map[group_name] = group
    for s in shapes:
        group.objects.link(shape(s, material_map))

for trf, group_name in world["instances"]:
    instance = bpy.data.objects.new(group_name, None)
    instance.instance_type = "COLLECTION"
    instance.instance_collection = collection_map[group_name]
    instance.matrix_world = mathutils.Matrix(trf)
```

Instance Conversion

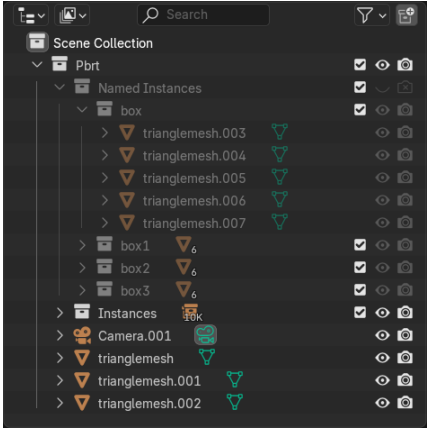


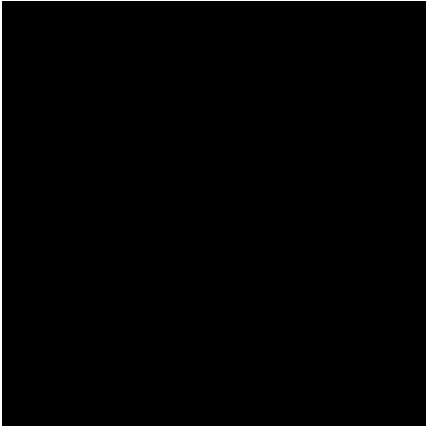
```
Transform [ 20 0 0 0 0 20 0 0 0 0 .01 0 0 0 -1 1 ]
Shape "trianglemesh" "integer indices" [ 0 1 2 2 3 0 ]
Shape "trianglemesh" "integer indices" [ 0 1 2 2 3 0 ]
Shape "trianglemesh" "integer indices" [ 0 1 2 2 3 0 ]
```



```
ObjectBegin "Mesh.001"
  Shape "trianglemesh" "integer indices" [ 0 1 2 2 3 0 ]
ObjectEnd
Transform [ 20 0 0 0 0 20 0 0 0 0 .01 0 0 0 -1 1 ]
ObjectInstance "Mesh.001"
ObjectInstance "Mesh.001"
ObjectInstance "Mesh.001"
```

Arranging the Scene in Collections — I





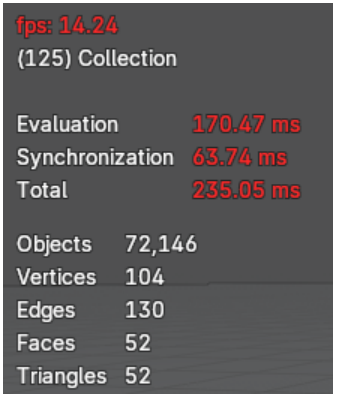
Arranging the Scene in Collections — II

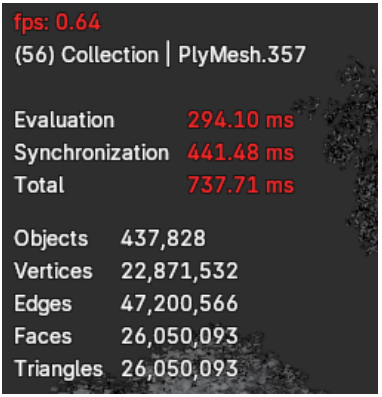



```
pbrt_col = bpy.data.collections.new("Pbrt")
bpy.context.scene.collection.children.link(pbrt_col)
instances_col = None
named_col = None
if world["named_objects"]:
    named_col = bpy.data.collections.new("Named Instances")
    pbrt_col.children.link(instance_col)
if world["instances"]:
    instance_col = bpy.data.collections.new("Instances")
    pbrt_col.children.link(named_instance_col)
```

Maintaining Interactivity — I

Working with Large Scenes





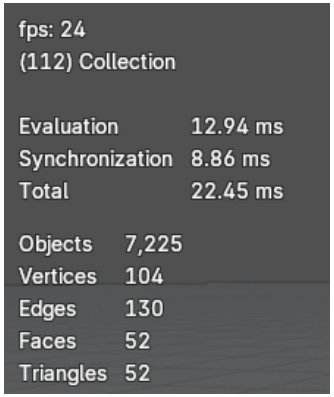
Maintaining Interactivity — II

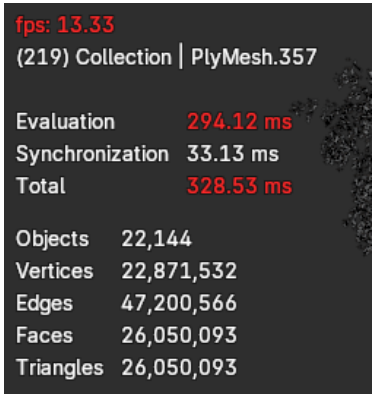
Working with Large Scenes

```
ins_ratio = settings.get("viewport_hiding", 0.0)
ins_counts = defaultdict(float)
ins_totals = defaultdict(int)
for trf, group_name in world["instances"]:
    ins_counts[group_name] += ins_ratio
    ins_totals[group_name] += 1
    cnt = ins_counts[group_name]
    tot = ins_totals[group_name]
    if tot > 1 and cnt >= 1.0:
        for vl in bpy.context.scene.view_layers:
            instance.hide_set(True, view_layer=vl)
        ins_counts[group_name] -= 1.0
```

Maintaining Interactivity — III

Working with Large Scenes

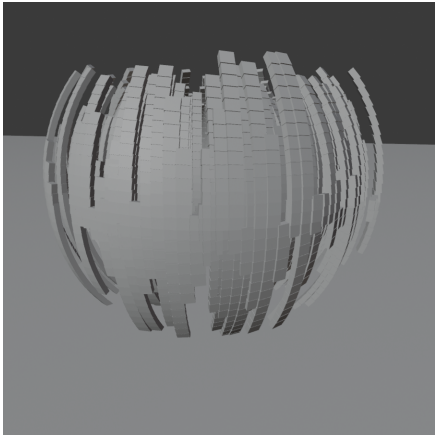


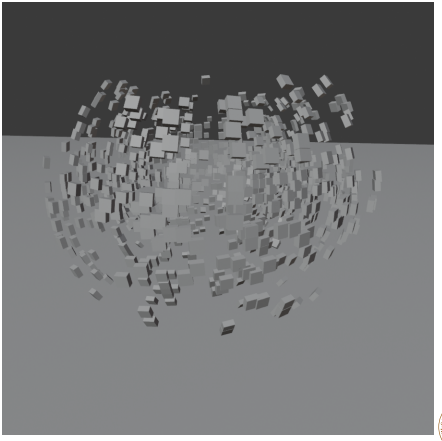


Hiding 90 % and 95 % of all instances respectively.

Maintaining Interactivity — IV

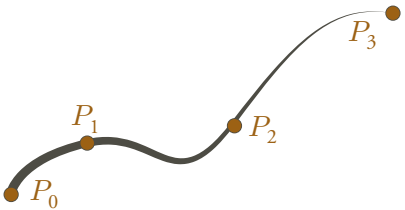
Working with Large Scenes



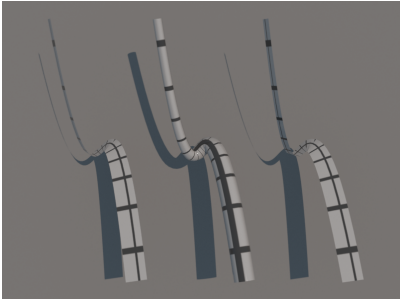


Hiding 90 %

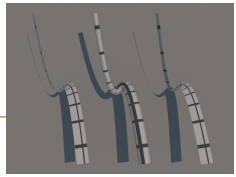
Curves

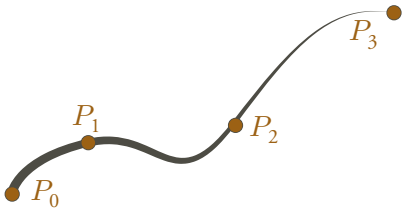



```
Shape "curve"
      "string type" "ribbon"
      "point P" [-2 -1 -1   2 2 0
                 -2 -2 0    2 1 -1]
      "normal N" [-.1 0 1 -.7 .1 .4]
      "float width0" [.35]
      "float width1" [.15]
```



Curves






```
op = bpy.ops.curve.primitive_nurbs_curve_add
obj = cached_creation("Curve", op)

spline = obj.data.splines[0]
spline.use_endpoint_u = True
spline.points[0].co.xyz = P[0]
spline.points[1].co.xyz = P[1]
spline.points[2].co.xyz = P[2]
spline.points[3].co.xyz = P[3]

spline.points[0].radius = width0 / 2
spline.points[1].radius = width0 / 2
spline.points[2].radius = width1 / 2
spline.points[3].radius = width1 / 2
```

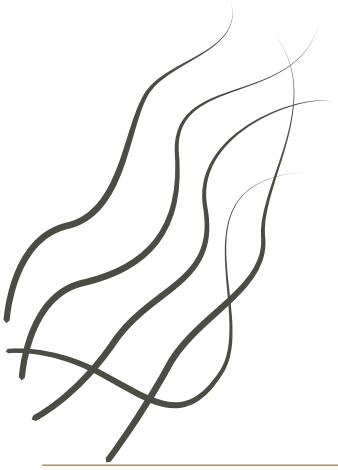
Curves — Hair

```
Shape "curve"
      "string type" ["cylinder"]
      "point P" [ -0.4 15.3 -1.0
                  -0.4 15.4 -1.0
                  -0.4 15.5 -1.0
                  -0.4 15.5 -1.0 ]
Shape "curve"
      "string type" ["cylinder"]
      "point P" [ -0.4 15.4 -1.0
                  -0.4 15.5 -1.0
                  -0.4 15.5 -1.0
                  -0.4 15.6 -1.1 ]
# ~3_000_000 more curves...
```



Curves — Bundling





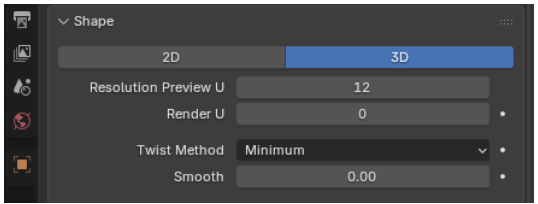
```
for c in curves:
    P = c.get("P", [])
    width0 = c.get("width0", [width])[0]
    width1 = c.get("width1", [width])[0]

    spline = obj.data.splines.new("NURBS")
    spline.points.add(3)

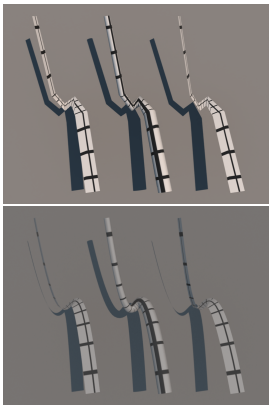
    spline.points[0].co.xyzw = (*P[0], 1.0)
    spline.points[1].co.xyzw = (*P[1], 1.0)
    spline.points[2].co.xyzw = (*P[2], 1.0)
    spline.points[3].co.xyzw = (*P[3], 1.0)

    spline.points[0].radius = width0 / 2
    spline.points[1].radius = width0 / 2
    spline.points[2].radius = width1 / 2
    spline.points[3].radius = width1 / 2
```

Curves — Resolution




```
obj.data.resolution_u = opt.get("curve_resolution_u", 1)
obj.data.resolution_v = opt.get("curve_resolution_v", 1)
```



Materials and Textures — I



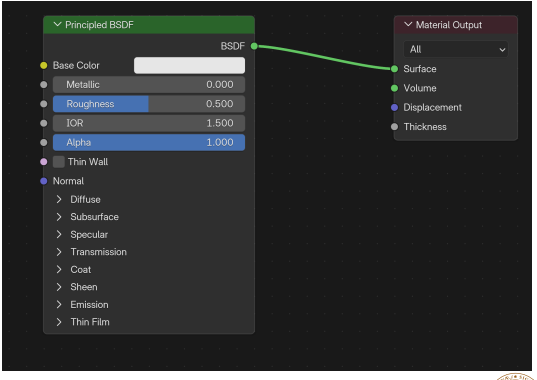
Materials and Textures — Building Graphs — II

```
bl_mat = bpy.data.materials.new(name="Uber")
bl_mat.use_nodes = True
bl_mat.node_tree.nodes.clear()

nodes = bl_mat.node_tree.nodes
links = bl_mat.node_tree.links

output = nodes.new("ShaderNodeOutputMaterial")
output.location = (400.0, 0.0)
bsdf = nodes.new("ShaderNodeBsdfPrincipled")
bsdf.location = (0.0, 0.0)

links.new(bsdf.outputs["BSDF"],
          output.inputs["Surface"])
```



Materials and Textures — III

PBRT

```
Texture "basecolor"
  "spectrum" "imagemap"
  "string filename"
  ["Abies_concolor_tr_01_co.png"]
Texture "bumpmap"
  "float" "imagemap"
  "string filename"
  ["Abies_concolor_tr_01_co.png"]
Material "uber"
  "float roughness" [ 0.50 ]
  "texture Kd" [ "basecolor" ]
  "texture bumpmap" [ "bumpmap" ]
  "rgb Kr" [ 0.033 0.033 0.033 ]
```

Python

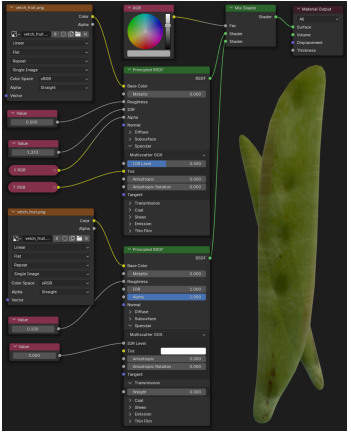
```
class Texture(typing.NamedTuple):
    value: str
    type: str
    parameters: dict

class Material(NamedTuple):
    type: str
    parameters: dict

world = {
    "materials": [...],
    "textures": [...],
}
```

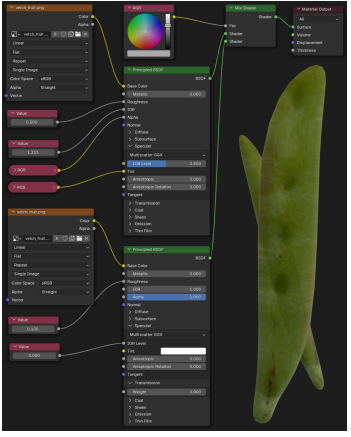
Materials and Textures — Socket Inputs — IV

- Scalars
- Colors
- Textures

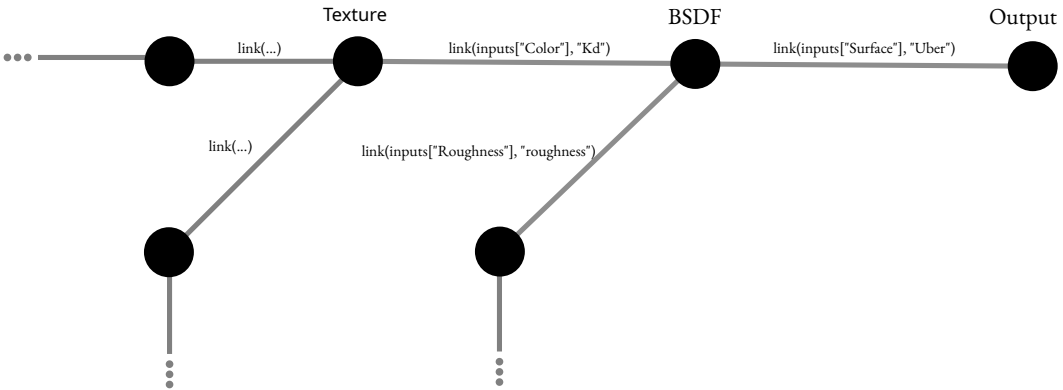


Materials and Textures — Socket Inputs — IV

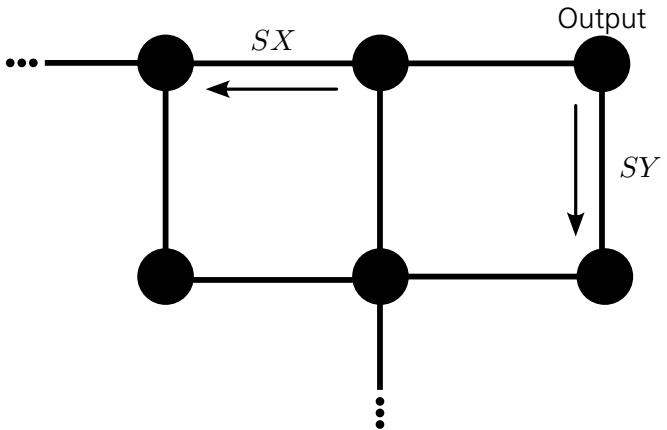
- Scalars
- Colors
- Textures
+ Mixing Textures
+ Mixing Materials



Materials and Textures — Recursion — V



Materials and Textures — Graphs Layout — VI



Materials and Textures — Assignment — VII

```
mat = material_map.get(s.material, material_map.get(0))

obj.data.materials.append(mat)
obj.active_material_index = len(obj.data.materials) - 1
obj.active_material = mat
```

Volumes

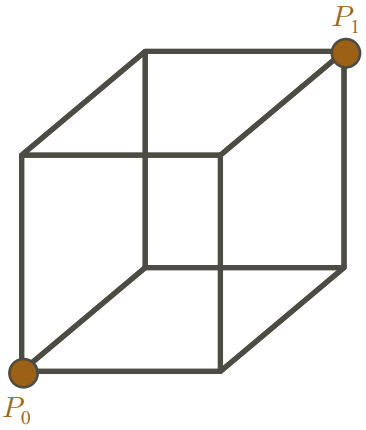


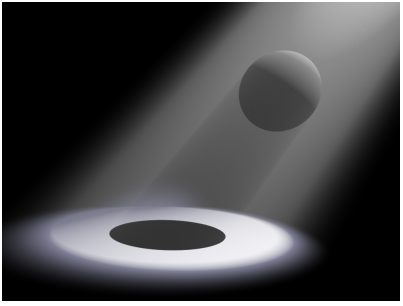
Homogeneous



Heterogeneous

Basic Homogeneous Volumes



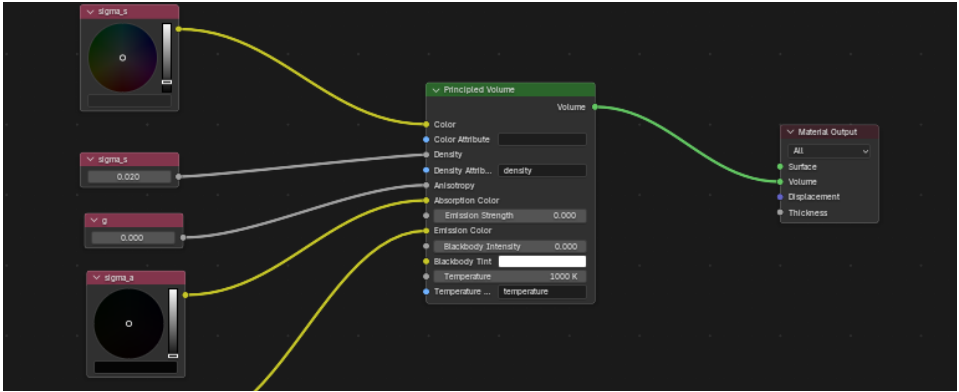


Homogeneous Medium Interfaces

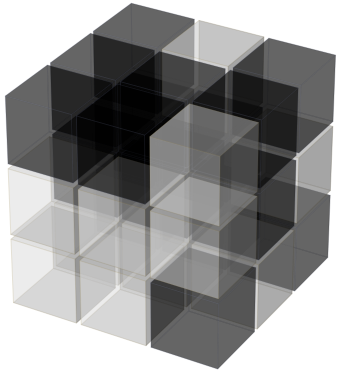
```
MakeNamedMedium "back"
                "string type" "homogeneous"
AttributeBegin
    MediumInterface "back" ""
    Material "glass"
    Shape "plymesh"
          "string filename" "dragon.ply"
AttributeEnd
```

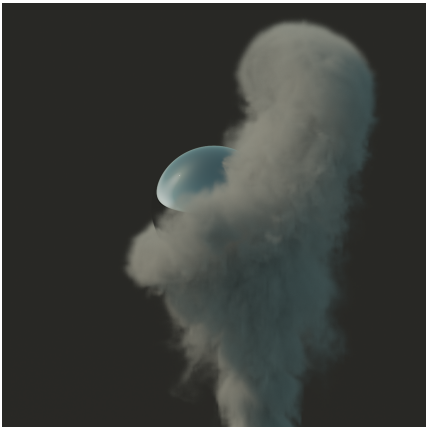


Volume Material



Hetereogeneous Media





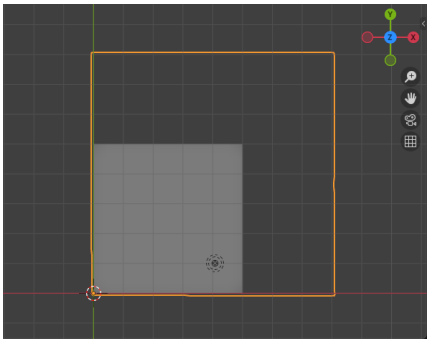
VDB Gotchas

Volumes — Voxel Grids

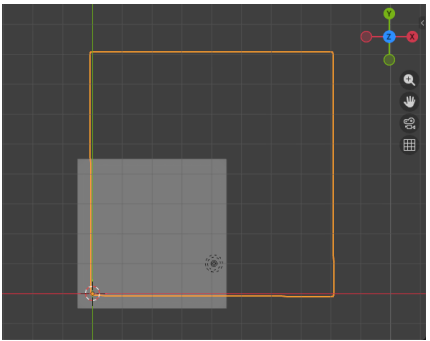
```
import openvdb
name = "density"
nx = ny = nz = 5
density = np.ones(shape=(nx, ny, nz))
path = bpy.path.abspath(f"//{name}.vdb")
grid = openvdb.FloatGrid()
grid.copyFromArray(density)
grid.name = name
openvdb.write(path, grids=[grid])
bpy.ops.object.volume_import(filepath=path)
```

VDB Gotchas

Volumes — Voxel Grids — Blender v4.4 vs Blender v5.1



v4.5



v5.1

VDB Gotchas

Volumes — Voxel Grids — Transforms

```
sx = 1.0 / nx
sy = 1.0 / ny
sz = 1.0 / nz
trf = openvdb.createLinearTransform([
    [sx,  0,   0,  0],
    [0,   sy,  0,  0],
    [0,    0, sz,  0],
    [0,    0,  0,  1],
])
grid.transform = trf
```

v4.5

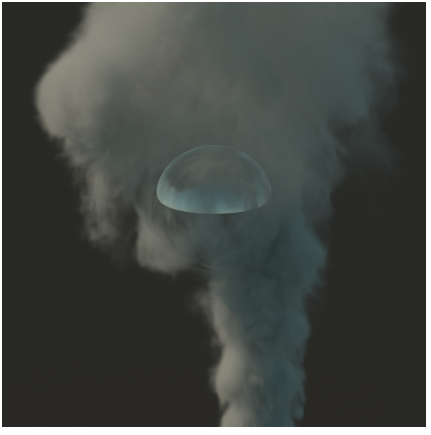
```
sx = 1.0 / nx
sy = 1.0 / ny
sz = 1.0 / nz
trf = openvdb.createLinearTransform([
    [sx,  0,   0,  0],
    [0,   sy,  0,  0],
    [0,    0, sz,  0],
    [sx / 2, sy / 2,  sz / 2,  1],
])
grid.transform = trf
```

v5.1

Volumes — Blender Labs

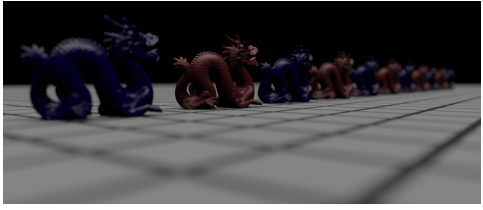
A Wish List?

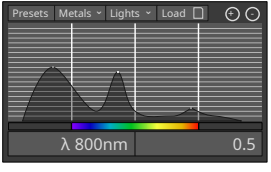
- In Memory Volumes
- Packing VDBs
- NanoVDB / NeuralVDB



Limitations

- Many materials/objects left to implement!
- Large scene support
- "Real-world properties"
  - lights and materials
  - Spectral values





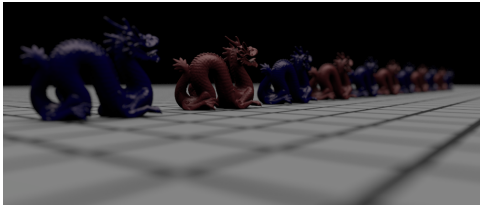
Large Scene Support

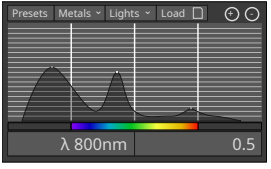
- Finding specific hidden objects?
- `material_utils` can help!



Real World Materials

- `Measured`
- `Fourier`
- Spectral Values

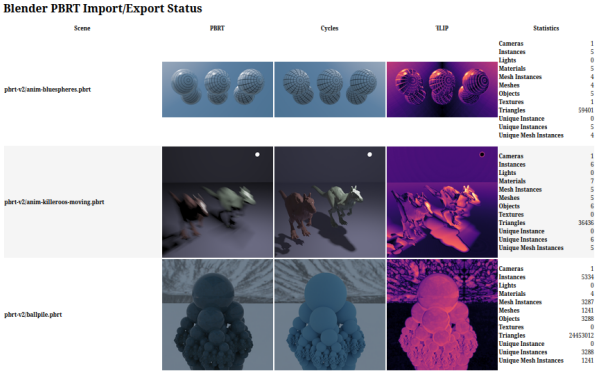




Distribution

- Codeberg



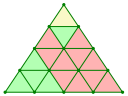
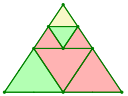
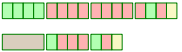


https://gustafwaldemarson.com/publications/blender-pbrt-import/
https://codeberg.org/Xaldew/blender-pbrt-io.git

Conclusions

1. PBRT and Cycles
2. How to create scenes
   - Curves
   - Materials
   - Volumes
3. Scaling it to Large Scenes with Interactivity
4. Limitations



PBRT 2: Creating and Working with Large Scenes

- `https://gustafwaldemarson.com/pages/publications/`
- gustaf.waldemarson@arm.com
- gustaf.waldemarson@gmail.com



The End

Extra Material



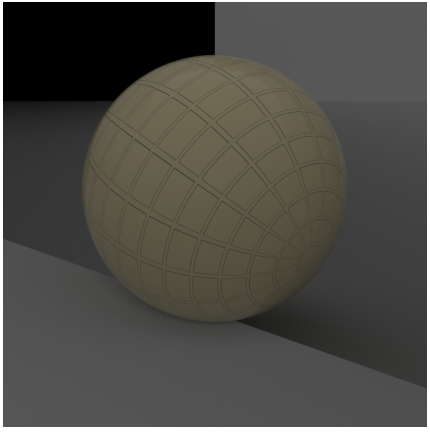
Triangle Meshes

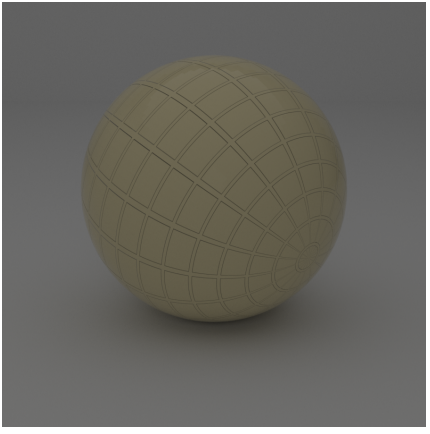
PLY Problems

```
#!/bin/sh
find "${1-.}" -type f -name "*.ply" \
     -exec sed -i 's/^property float u$/property float s/g' {} \; \
     -exec sed -i 's/^property float v$/property float t/g' {} \;
```

Triangle Meshes

Winding Woes



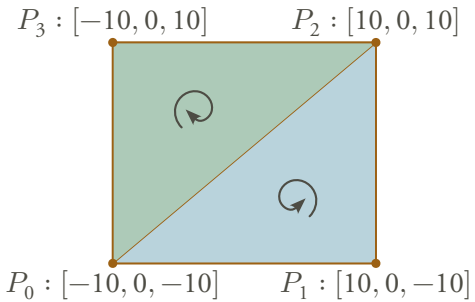


Triangle Meshes

Winding Woes

```
AttributeBegin
  Shape "trianglemesh"
        "integer indices"  [0 1 2  2 0 3]
        "point P" [-10 0 -10   10 0 -10
                   10 0 10    -10 0  10 ]
AttributeEnd
```

```
AttributeBegin
  Shape "trianglemesh"
        "integer indices"  [0 1 2  2 3 0]
        "point P" [-10 0 -10   10 0 -10
                   10 0 10    -10 0  10 ]
AttributeEnd
```



References I

[1] Pontus Andersson, Jim Nilsson, and Tomas Akenine-Möller. "Visualizing and Communicating Errors in Rendered Images". In: *Ray Tracing Gems II*. Ed. by Adam Marrs, Peter Shirley, and Ingo Wald. 2021. Chap. 19, pp. 301–320.

[2] Matt Pharr, Greg Humphreys, and Wenzel Jakob. *Physically Based Rendering: From Theory to Implementation*. 1st ed. 2004; 2nd ed. 2010; 3rd ed. 2016; 4th ed. 2023. Online 3rd and 4th ed. at `https://pbr-book.org/3ed-2018` and `https://pbr-book.org/4ed`. Morgan Kaufmann and MIT Press, 2023. URL: `https://pbr-book.org`.

[3] Gustaf Waldemarson. *PBRT 2: Creating and Working with Large Scenes*. The Blender Foundation, Youtube. 2026. URL: `https://youtu.be/iao8H8jPrqA`.

[4] Gustaf Waldemarson. *PBRT: Create your own Importers and Exporters*. The Blender Foundation, Youtube. 2024. URL: `https://youtu.be/BEbscsBRIx0`.