Physically Based Rendering

...

An advanced practicum by Michael Pronkin

Overview

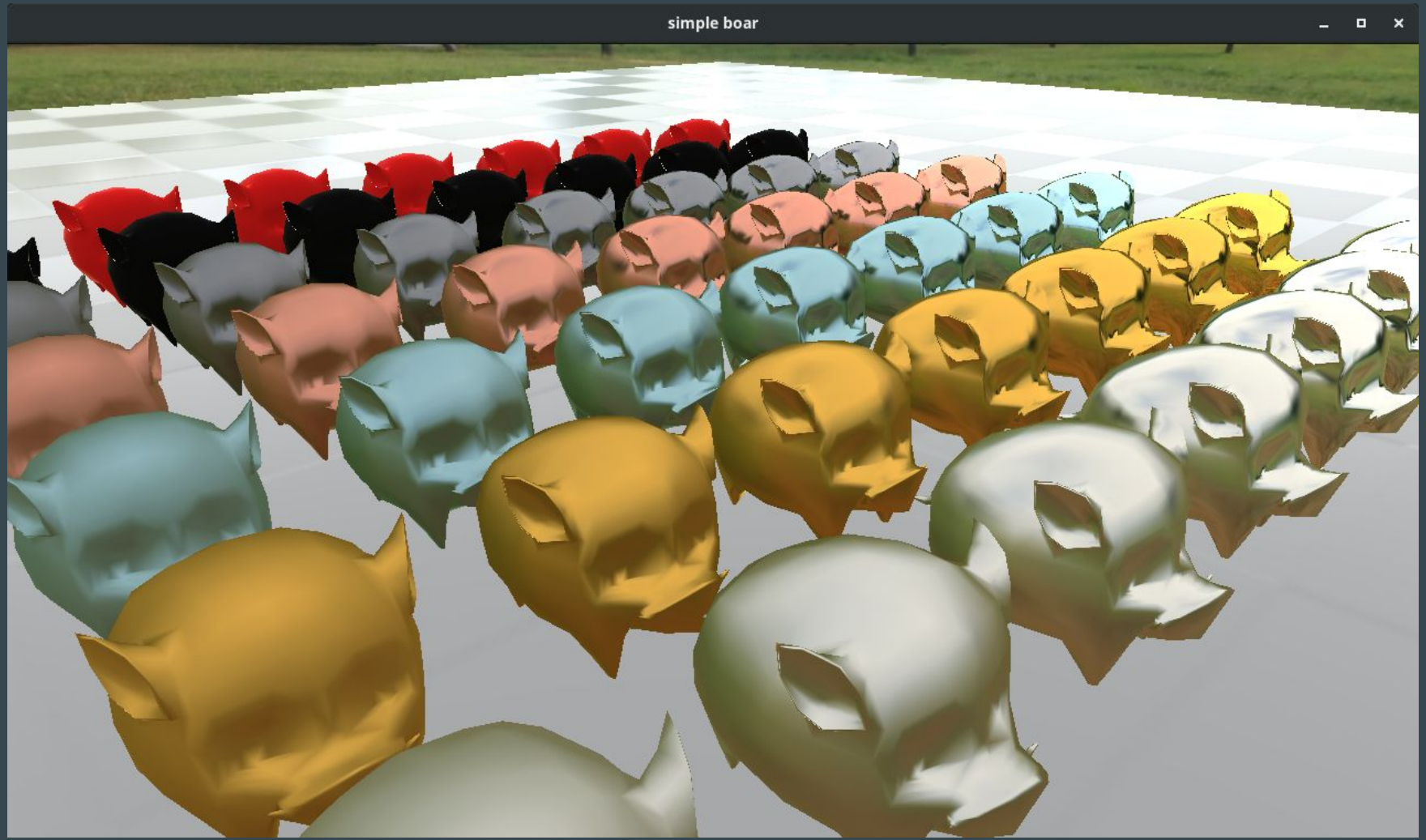
- PBR
  - intro
  - BRDF (bidirectional reflectance distribution function)
  - materials
  - IBL (image based lighting)
- implementation
  - Deferred Rendering
  - GBuffer for PBR
  - material data structure
- outlook
  - gltf
  - PBR in the industry

Intro to PBR
(Physically Based Rendering)

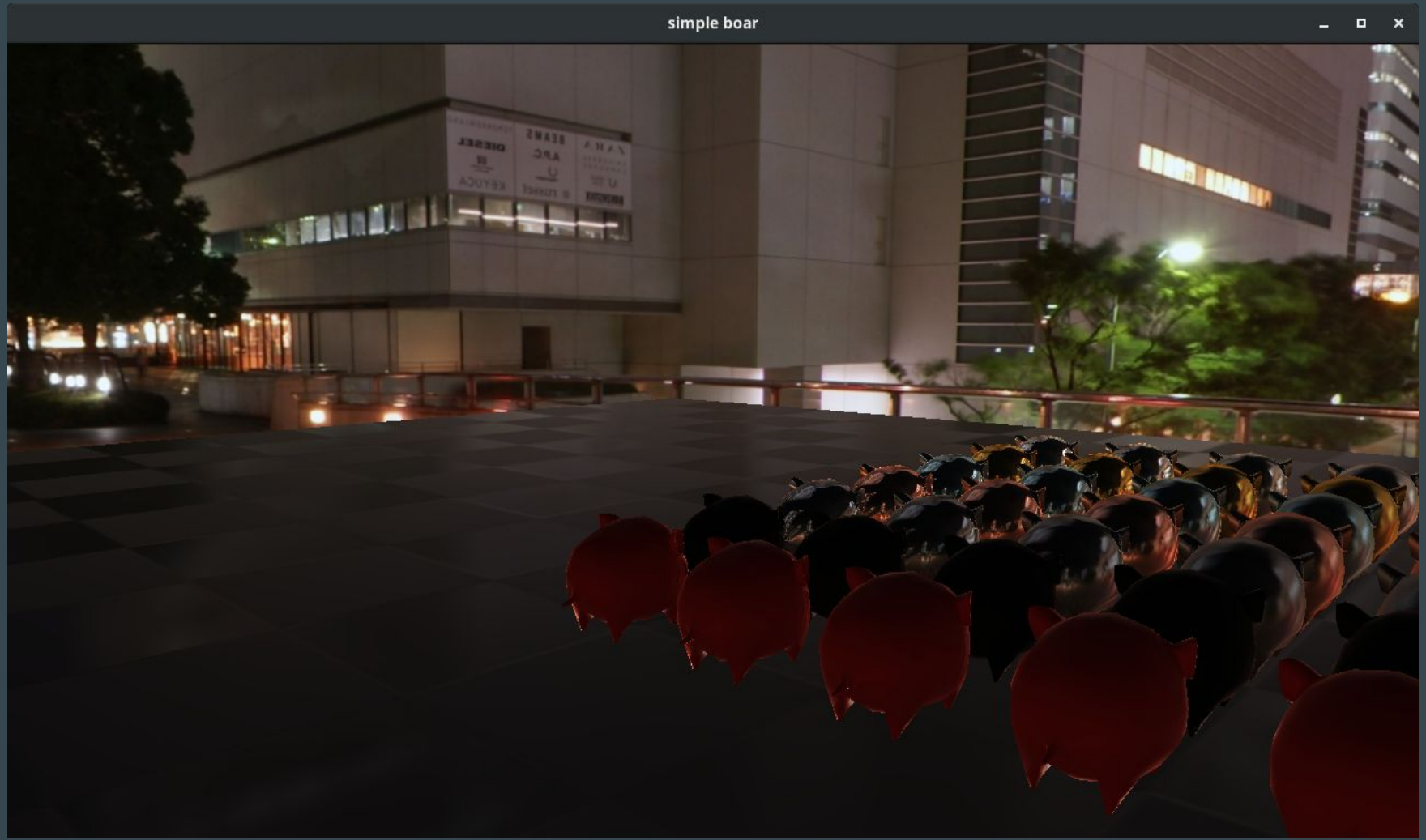
material layering

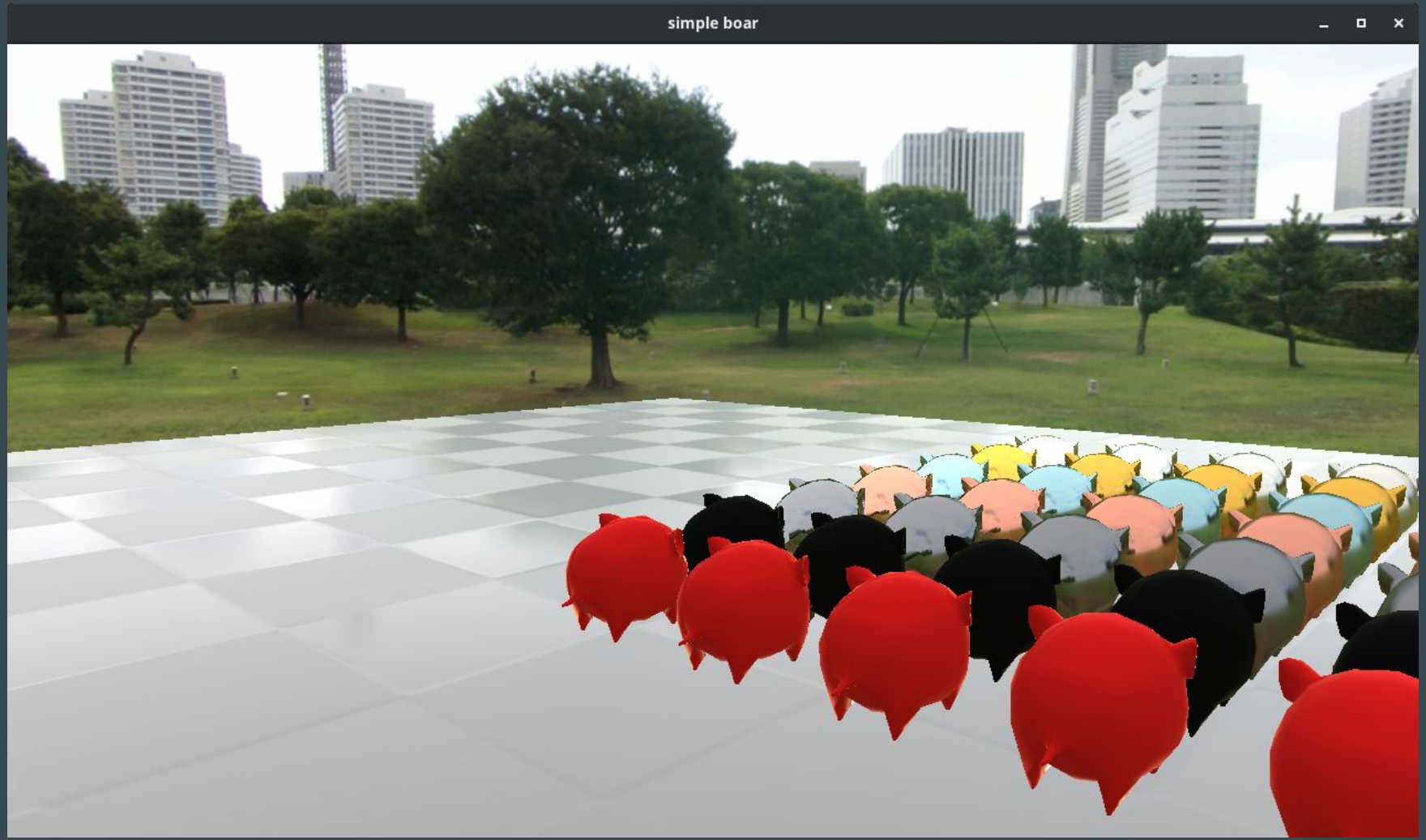
From Marmoset Toolbag Tutorials: Physically-Based Rendering, And You Can Too!, by Joe "Earthquake" Wilson

From Marmoset Toolbag Tutorials: Physically-Based Rendering, And You Can Too!, by Joe "Earthquake" Wilson


simple boar

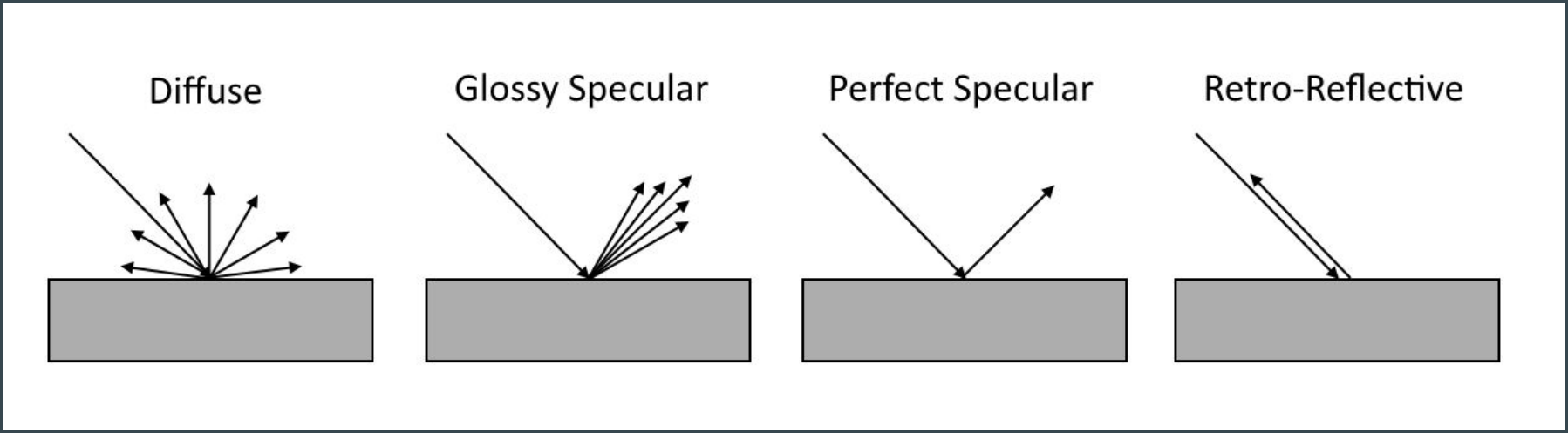

simple boar


simple boar


simple boar

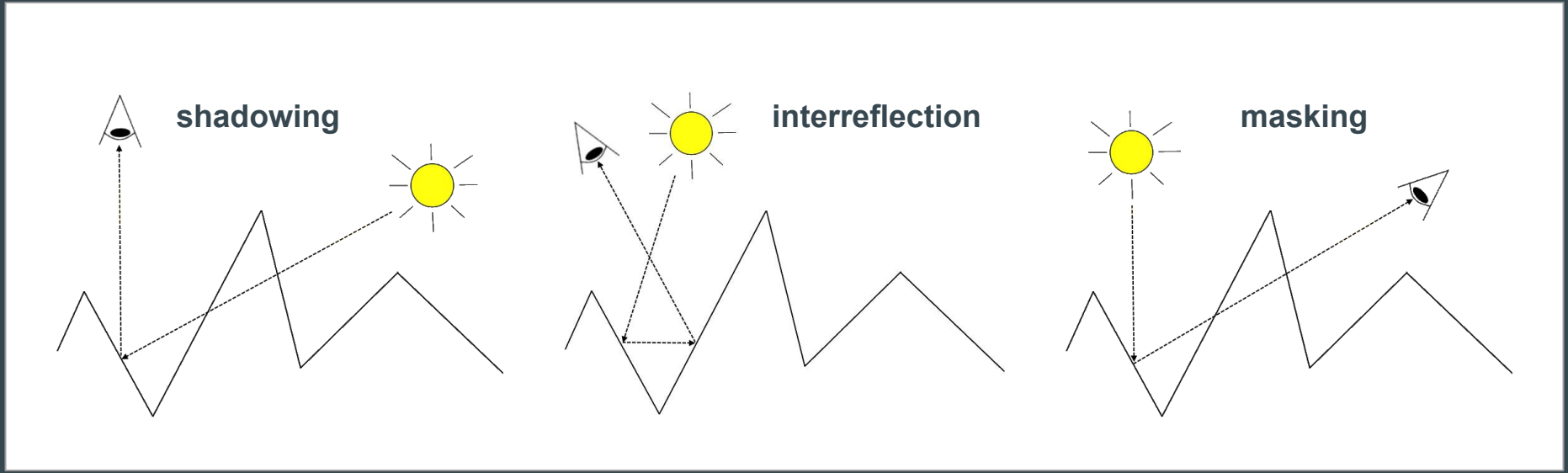
BRDF (bidirectional reflectance distribution function)

Describes **reflection** of light from a point on a surface



From https://github.com/moneimne/glTF-Tutorials/tree/master/PBR/figures

Geometric Occlusion



From https://github.com/moneimne/glTF-Tutorials/tree/master/PBR/figures

BRDF: Lighting Function

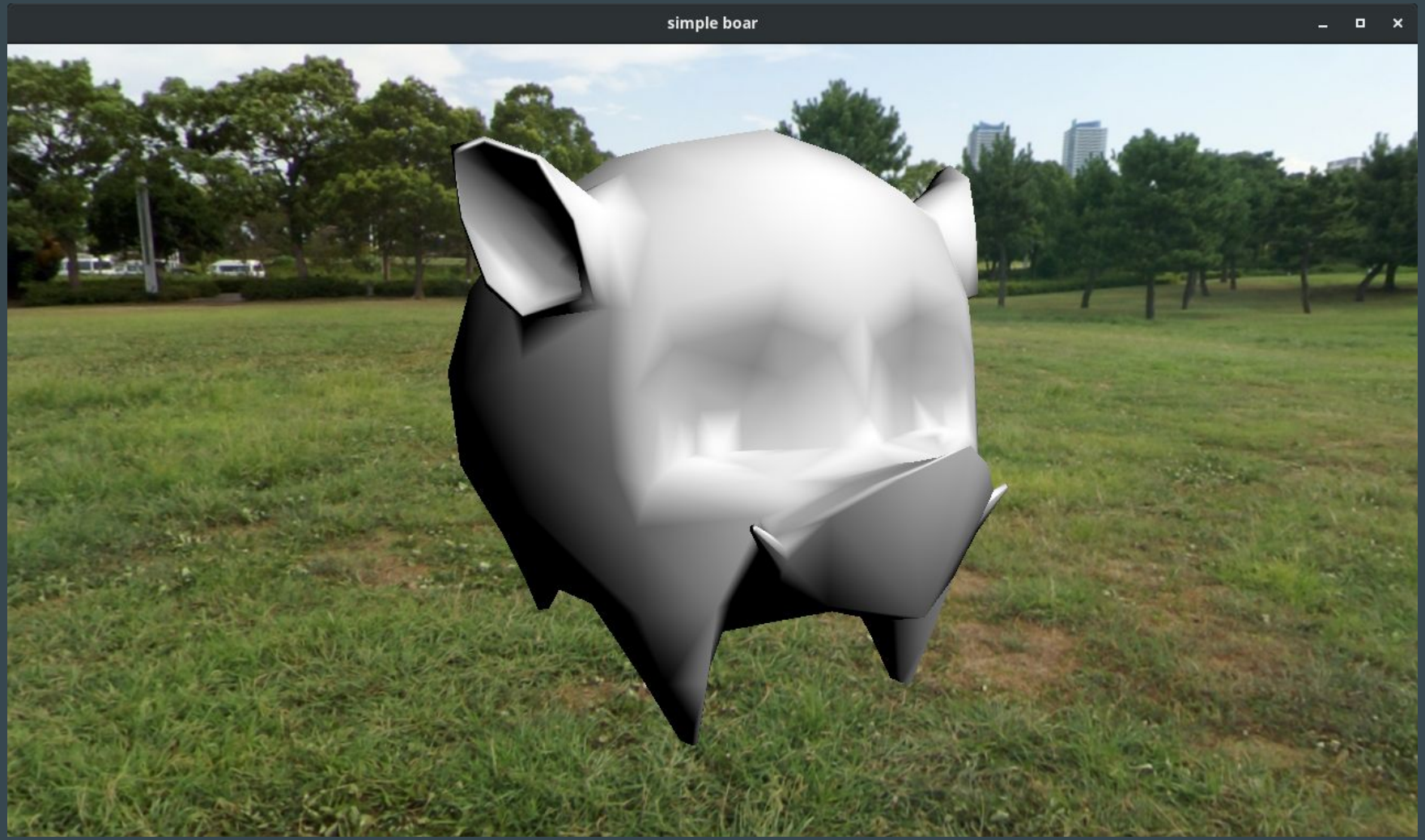
$$f(l, v, n) = \underbrace{Diffuse(l, n)}_{\text{Diffuse Term (Lambert)}} + \underbrace{\frac{F(l, h)\, G(l, v, h)\, D(h)}{4(n \cdot l)(n \cdot v)}}_{\text{Specular Term (Cook-Torrance)}}$$

- *l := light direction*
- *v := view direction*
- *h := light view halfway vector*
- *n := normal vector*

BRDF Diffuse Term (Lambert)

$$Diffuse(l, n) = l \cdot n$$

- l := *light direction*
- n := *normal vector*


simple boar

BRDF Specular Term (Cook-Torrance)

$$\frac{F(l, h)\, G(l, v, h)\, D(h)}{4(n \cdot l)(n \cdot v)}$$

- l := *light direction*
- v := *view direction*
- h := *light view halfway vector*
- n := *normal vector*

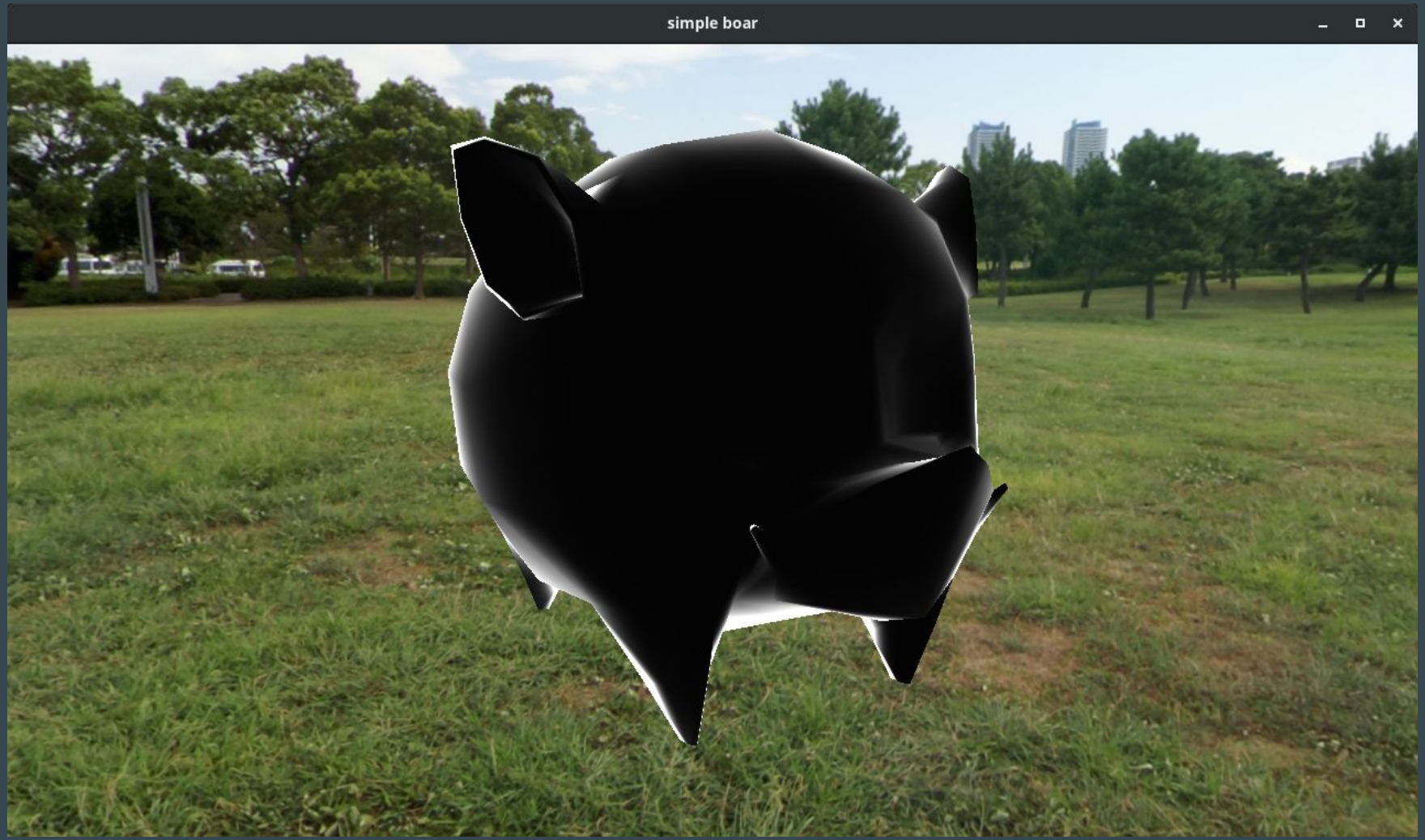
- ***F**resnel Term*
- ***G**eometric Occlusion Term*
- ***D**istribution Term*

BRDF Specular Term (Fresnel)

$$\frac{F(l, h)\, G(l, v, h)\, D(h)}{4(n \cdot l)(n \cdot v)}$$

$$F_{schlick\_gauss}(l, h) = F_0 + (1 - F_0) \times (1 - v \cdot h)^5$$

- F_0 := *specular reflectance at normal incidence*
- l := *light direction*
- h := *light view halfway vector*


simple boar

BRDF Specular Term (Geometric Occlusion)

$$\frac{F(l, h)\, G(l, v, h)\, D(h)}{4(n \cdot l)(n \cdot v)}$$

$$G_{schlick}(l, v, h) = G_1(n, l) \times G_1(n, v)$$

$$G_1(n, v) = \frac{2 \times (n \cdot v)}{(n \cdot v) + \sqrt{r^2 + (1 - r^2)(n \cdot v)^2}}$$

- r := *roughness*
- l := *light direction*
- v := *view direction*
- h := *light view halfway vector*

BRDF Specular Term (Distribution)

$$\frac{F(l, h)\, G(l, v, h)\, D(h)}{4(n \cdot l)(n \cdot v)}$$

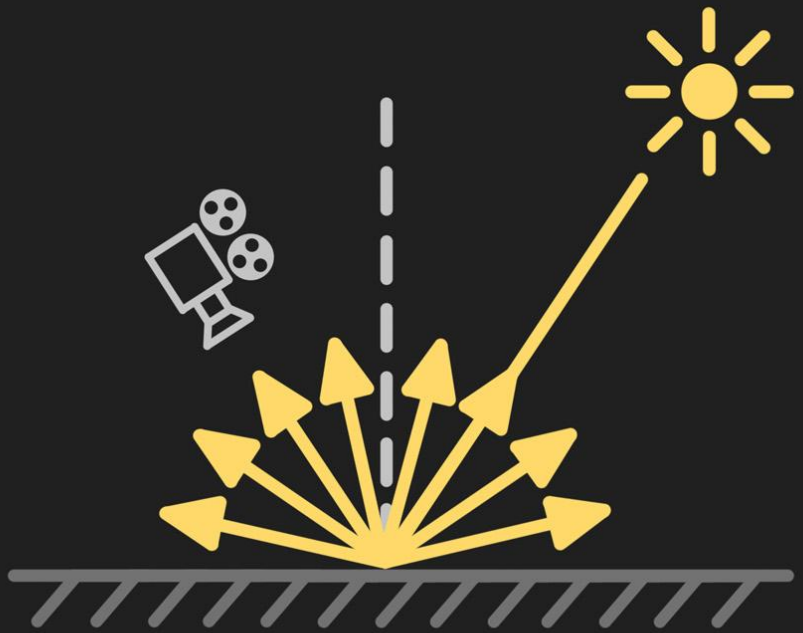
$$D_{ggx}(h) = \frac{r^4}{\pi((n \cdot h)^2 \times (r^4 - 1) + 1)^2}$$

- r := roughness
- h := is actually $n \cdot h$ for this term

IBL (image based lighting)

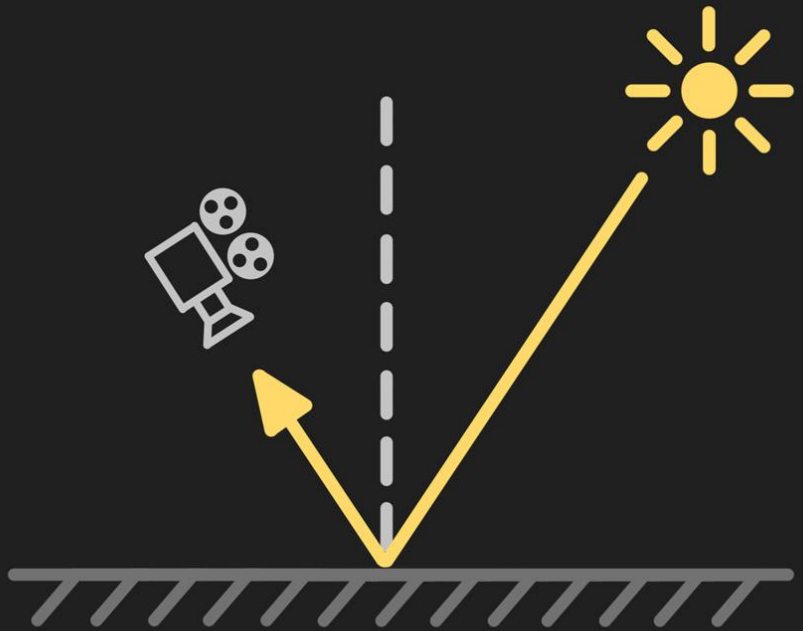
- cubemap (or similar)
- multiple resolutions
- for both diffuse and specular
- point/area lights optional
- HDR (high dynamic range)

DIFFUSE



from https://3dcoat.com/pbr/

SPECULAR



from https://3dcoat.com/pbr/

implementation

- BRDF code
- Deferred Rendering
- GBuffer for PBR
- irradiance/reflection maps
- material data structure
  - glTF format

BRDF code

```
float brdf_diffuse_lambert(vec3 light_dir, vec3 normal) {
    return max(dot(light_dir, normal), 0.0f);
}
```

```
float brdf_specular_cook_torrance(vec3 light_dir, vec3 view_dir, vec3 normal, float rough, float metal) {
    vec3 halfway_vector = normalize(light_dir + view_dir);
    float normal_dot_light = dot(normal, light_dir);
    float normal_dot_view = dot(normal, view_dir);
    float normal_dot_halfway = dot(normal, halfway_vector);
    float view_dot_halfway = dot(view_dir, halfway_vector);

    float D = distribution_ggx(normal_dot_halfway, rough);
    float F = fresnel_schlick_gauss(view_dot_halfway, metal);
    float G = geometry_schlick(normal_dot_light, normal_dot_view, rough);

    return D * F * G / (4.0f * normal_dot_light * normal_dot_view);
}
```

Deferred Rendering vs. Forward Rendering

- Lights per fragment
  (less complexity)
- Only calculates visible pixels
  (screen space deferral)
- Requires buffers
  (multi render target)
- Anti-aliasing more difficult

- Lights per vertex
- Requires additional calculations
  for invisible geometry avoidance
  (or naively calculate all invisible)
- Much lower overhead in memory

Gbuffer for PBR

- albedo
- normal/roughness
- specular/IOR
- position/depth

Gbuffer code

```
class GBuffer : public FrameBuffer {
public:
    GBuffer(
        std::shared_ptr<TextureAsset> albedo_tex          = std::make_shared<TextureAsset>(TextureFormat::default_rgba()),
        std::shared_ptr<TextureAsset> normal_rough_tex    = std::make_shared<TextureAsset>(TextureFormat::default_rgba16f()),
        std::shared_ptr<TextureAsset> specular_reflect_tex = std::make_shared<TextureAsset>(TextureFormat::default_rgba16f()),
        std::shared_ptr<TextureAsset> pos_depth_tex       = std::make_shared<TextureAsset>(TextureFormat::default_rgba16f()),
        std::shared_ptr<TextureAsset> depth_tex           = std::make_shared<TextureAsset>(TextureFormat::default_depth()));
    void draw(ShaderProgram& shader) const;
    void resize(glm::vec2 to);

protected:
    std::shared_ptr<TextureAsset> normal_rough_tex;
    std::shared_ptr<TextureAsset> specular_tex;
    std::shared_ptr<TextureAsset> pos_depth_tex;
};

```

Gbuffer code continued

```
sbg::GBuffer g_buffer{
    a_window.asset_manager.get_texture_asset("g_albedo",          sbg::TextureFormat::default_rgba()),
    a_window.asset_manager.get_texture_asset("g_normal_rough",    sbg::TextureFormat::default_rgba16f()),
    a_window.asset_manager.get_texture_asset("g_specular_reflect", sbg::TextureFormat::default_rgba16f()),
    a_window.asset_manager.get_texture_asset("g_pos_depth",       sbg::TextureFormat::default_rgba16f()),
    a_window.asset_manager.get_texture_asset("g_depth",           sbg::TextureFormat::default_depth())};

```

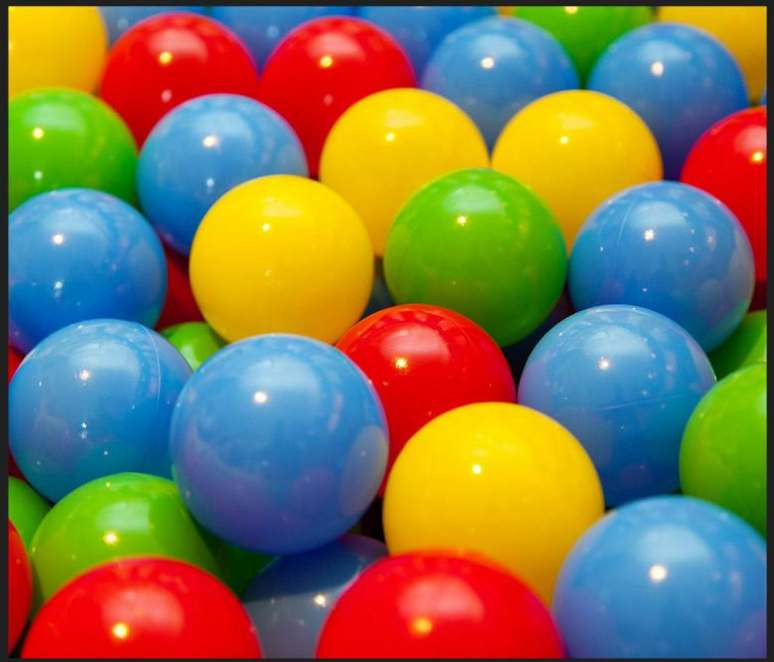
```
g_buffer.bind();
gbuffer_shader.bind();

// draw scene

glBindFramebuffer(GL_FRAMEBUFFER, 0); // render to screen
g_buffer.draw(deferred_shader);

```

materials

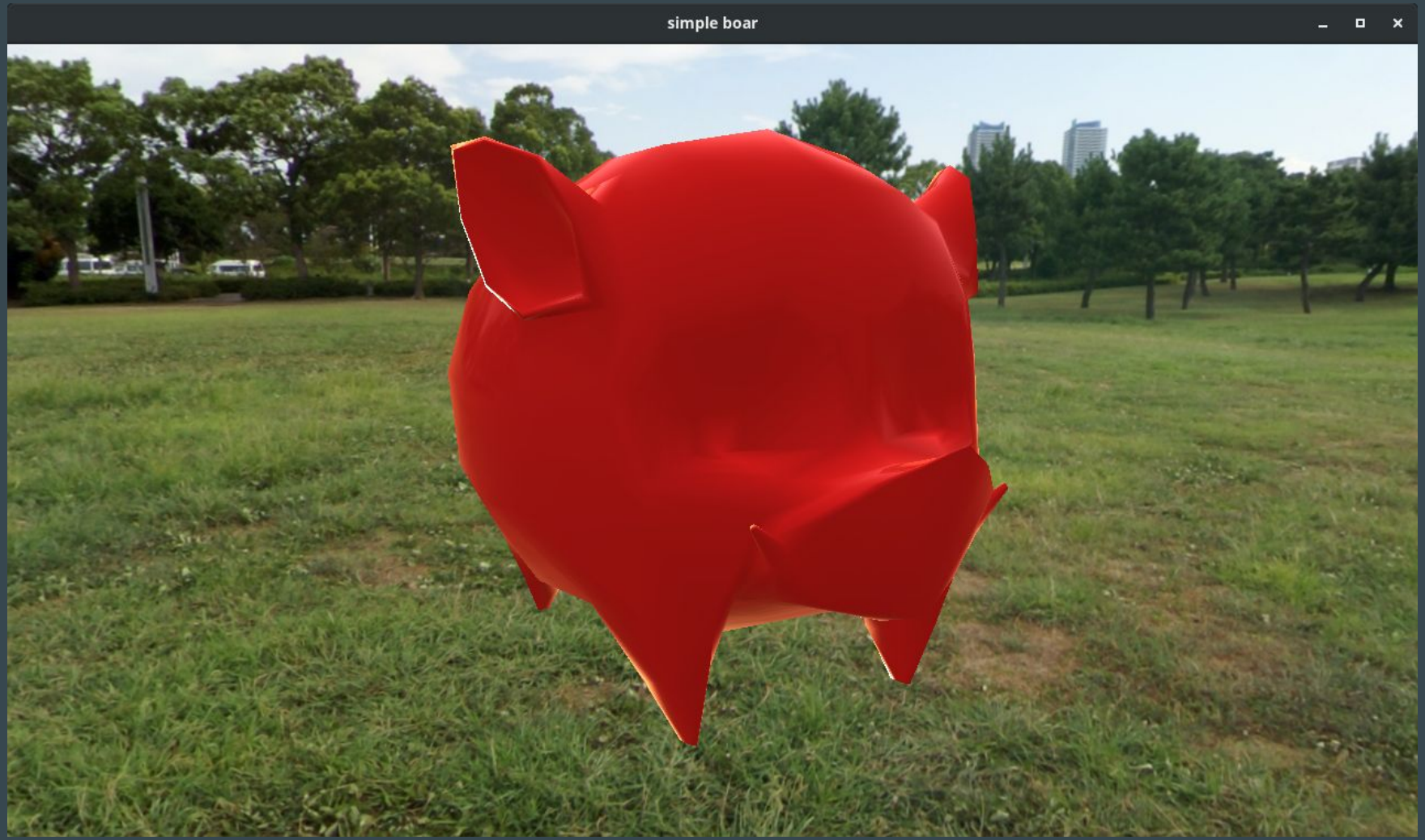
components:

- albedo (“color”)
- metalness (“metal or dielectric”)
- roughness (“rough to smooth”)

DIELECTRICS

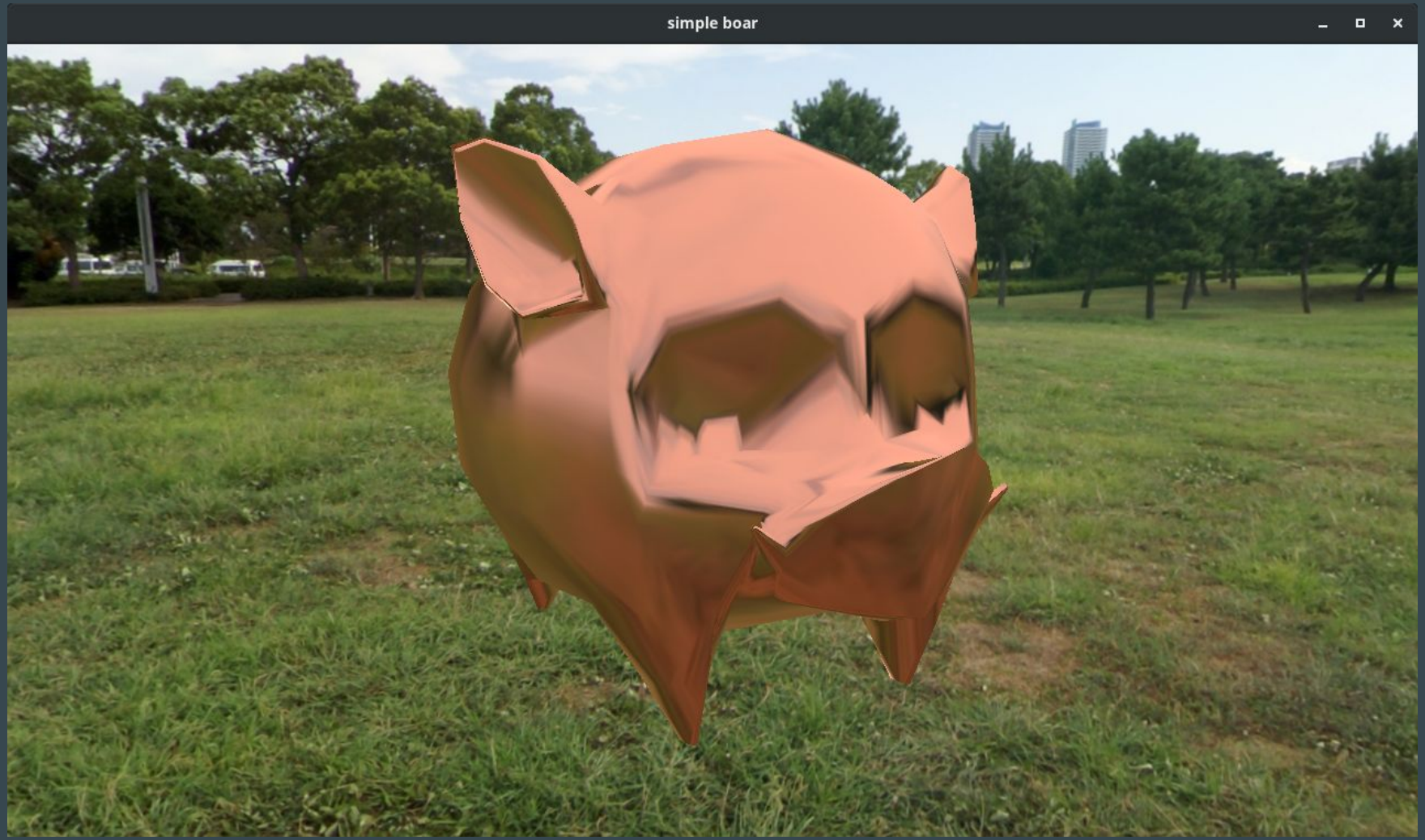


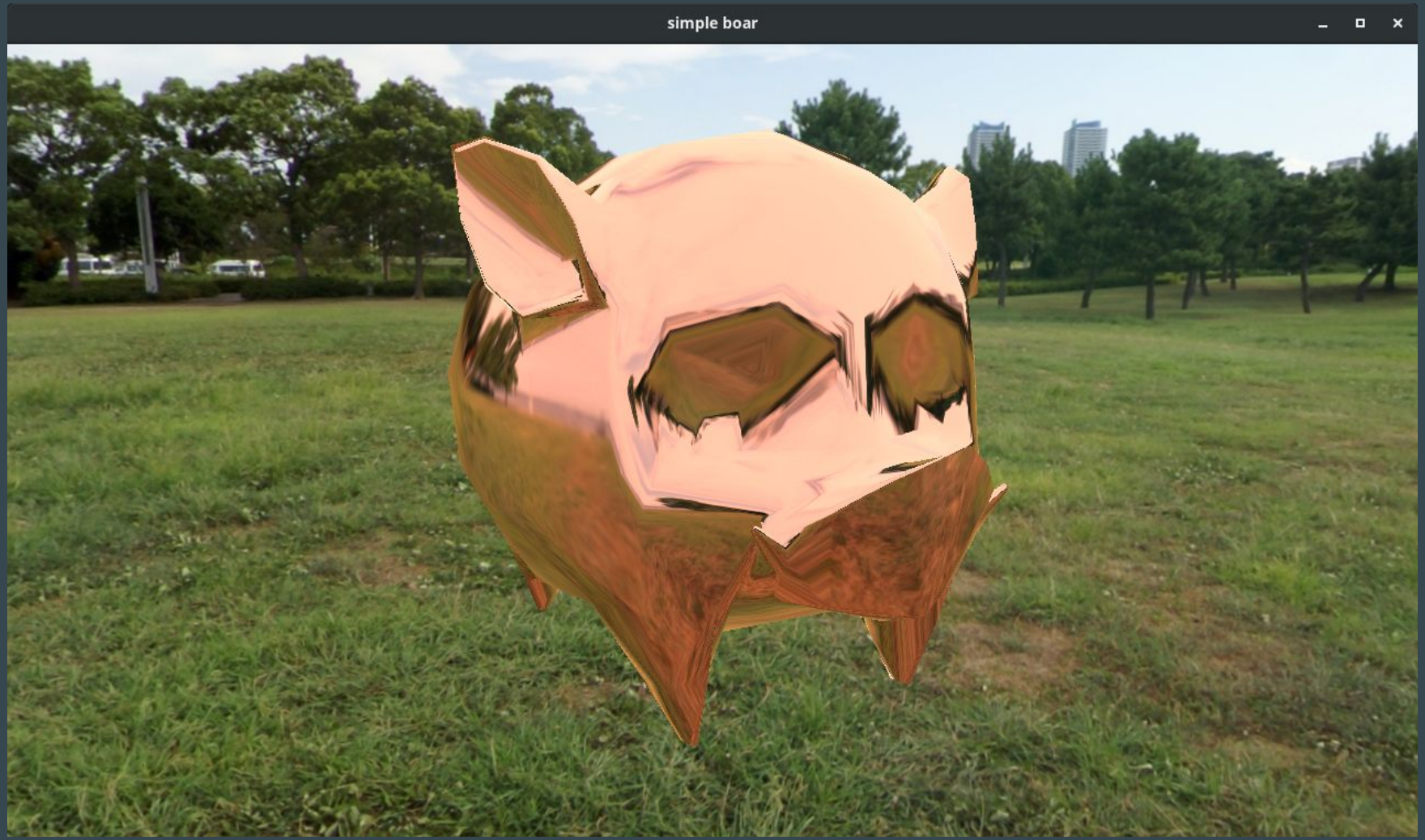
from https://3dcoat.com/pbr/


simple boar

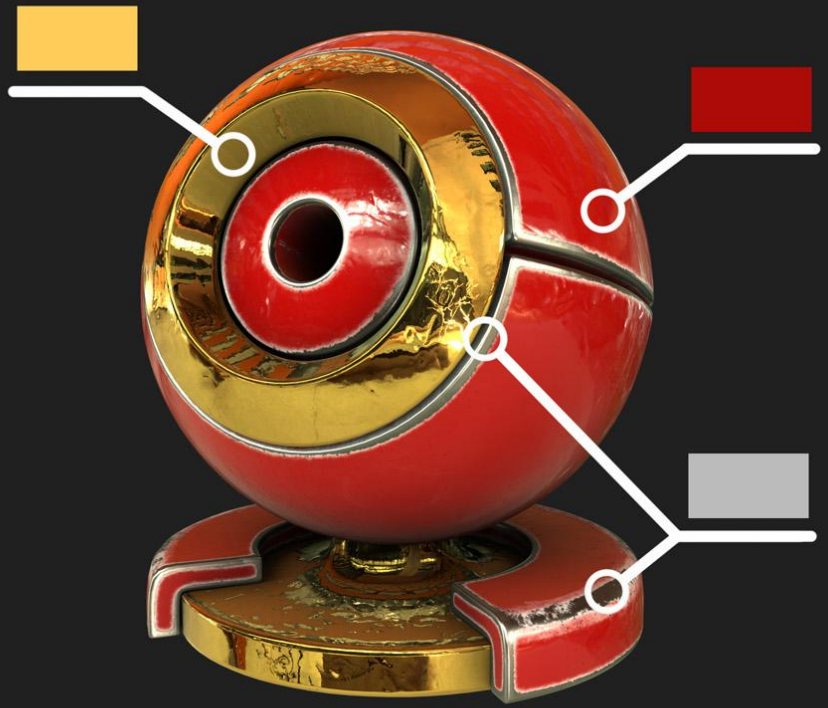
METALS

from https://3dcoat.com/pbr/


simple boar

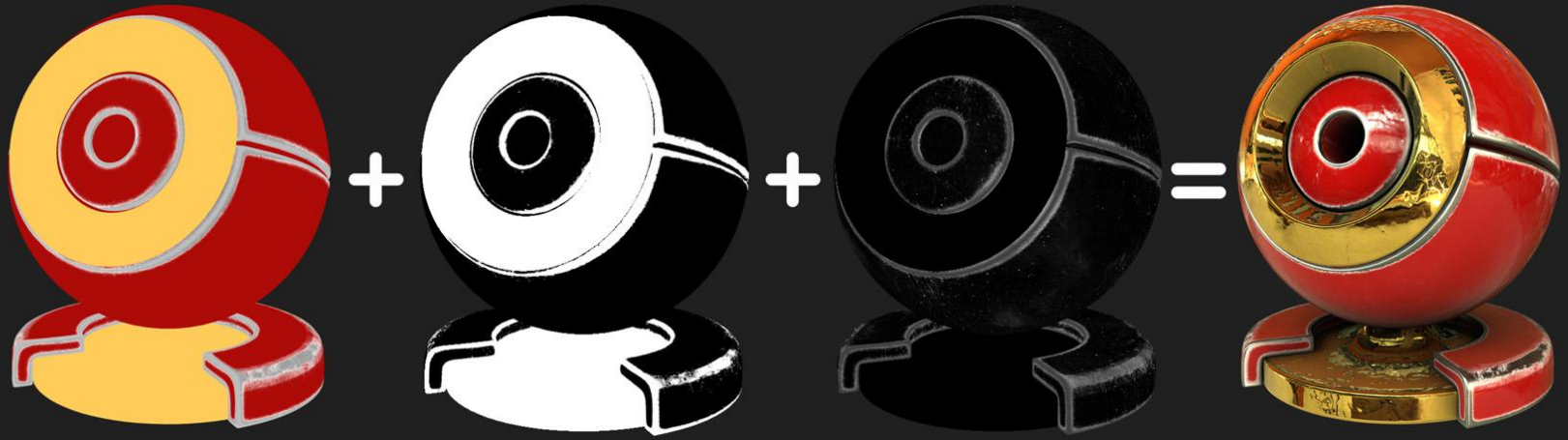

simple boar

ALBEDO



from https://3dcoat.com/pbr/

TEXTURES FOR METALNESS WORKFLOW



from https://3dcoat.com/pbr/

PBR vs. traditional shading

- higher memory overhead
- much higher visual impact
- closer approximation of photorealism

- Simple to understand and implement
- useful for prototyping
- easy to run on very old hardware

glTF

- New format for PBR based scenes
- Application independent
- compact size
- fast loading
- open and extensible
- you can follow all development at
  https://github.com/KhronosGroup/glTF

PBR in the industry

demo time!

thanks!