Path tracing 1.0

Generated by Doxygen 1.10.0

1	Class Index	1
	1.1 Class List	1
2	File Index	3
	2.1 File List	3
3	Class Documentation	5
	3.1 AABB Struct Reference	5
	3.1.1 Detailed Description	5
	3.1.2 Constructor & Destructor Documentation	5
	3.1.2.1 AABB() [1/2]	5
	<b>3.1.2.2 AABB()</b> [2/2]	6
	3.1.3 Member Function Documentation	6
	3.1.3.1 Empty()	6
	3.1.3.2 GetSurfaceArea()	6
	3.1.3.3 IntersectsRay()	6
	3.1.4 Member Data Documentation	7
	3.1.4.1 max	7
	3.1.4.2 min	7
	3.2 AccelerationStructure Class Reference	7
	3.2.1 Detailed Description	7
	3.2.2 Constructor & Destructor Documentation	7
	3.2.2.1 AccelerationStructure() [1/2]	7
	3.2.2.2 AccelerationStructure() [2/2]	7
	3.2.2.3 ~AccelerationStructure()	8
	3.2.3 Member Function Documentation	8
	3.2.3.1 Hit()	8
	3.2.3.2 Update()	8
	3.3 Application Class Reference	8
	3.3.1 Detailed Description	9
	3.3.2 Constructor & Destructor Documentation	9
	3.3.2.1 Application()	9
	3.3.3 Member Function Documentation	10
	3.3.3.1 Run()	10
	3.4 BSDF Class Reference	10
	3.4.1 Detailed Description	11
	3.4.2 Constructor & Destructor Documentation	11
	3.4.2.1 BSDF()	11
	3.4.3 Member Function Documentation	11
	3.4.3.1 Sample()	11
	3.5 BVHNode Struct Reference	13
	3.5.1 Detailed Description	13
	3.5.2 Constructor & Destructor Documentation	14
	5.5.2 55.36 dots & Book dots Book market 1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.	, ,

3.5.2.1 BVHNode() [1/2]	 . 14
3.5.2.2 BVHNode() [2/2]	 . 14
3.5.3 Member Function Documentation	 . 14
3.5.3.1 FreeMemory()	 . 14
3.5.3.2 Hit()	 . 14
3.5.3.3 IsTerminating()	 . 15
3.5.3.4 MakeHierarchyNaive()	 . 15
3.5.3.5 MakeHierarchySAH()	 . 15
3.5.4 Member Data Documentation	 . 16
3.5.4.1 aabb	 . 16
3.5.4.2 c_ComparatorsNaive	 . 16
3.5.4.3 c_ComparatorsSAH	 . 17
3.5.4.4 left	 . 17
3.5.4.5 object	 . 17
3.5.4.6 right	 . 17
3.6 Camera Class Reference	 . 18
3.6.1 Detailed Description	 . 18
3.6.2 Constructor & Destructor Documentation	 . 18
3.6.2.1 Camera()	 . 18
3.6.3 Member Function Documentation	 . 19
3.6.3.1 ComputeRayDirections()	 . 19
3.6.3.2 GetPosition()	 . 19
3.6.3.3 GetRayDirections()	 . 19
3.6.3.4 GetTarget()	 . 19
3.6.3.5 GetUp()	 . 20
3.6.3.6 GetVerticalFovInDegrees()	 . 20
3.6.3.7 OnViewportResize()	 . 20
3.6.3.8 Position()	 . 20
3.6.3.9 Target()	 . 20
3.6.3.10 Up()	 . 21
3.6.3.11 VerticalFovInDegrees()	 . 21
3.7 HitPayload Struct Reference	 . 21
3.7.1 Detailed Description	 . 21
3.7.2 Member Data Documentation	 . 21
3.7.2.1 material	 . 21
3.7.2.2 normal	 . 21
3.7.2.3 t	 . 22
3.8 Image Class Reference	 . 22
3.8.1 Detailed Description	 . 22
3.8.2 Constructor & Destructor Documentation	 . 22
3.8.2.1 Image()	 . 22
3.8.2.2 ∼Image()	 . 22

3.8.3 Member Function Documentation	23
3.8.3.1 GetDescriptor()	23
3.8.3.2 UpdateData()	23
3.9 Light Class Reference	23
3.9.1 Detailed Description	23
3.9.2 Constructor & Destructor Documentation	23
3.9.2.1 Light()	23
3.9.3 Member Function Documentation	24
3.9.3.1 GetObject()	24
3.9.3.2 Sample()	24
3.10 Material Struct Reference	24
3.10.1 Detailed Description	25
3.10.2 Member Function Documentation	25
3.10.2.1 GetEmission()	25
3.10.3 Member Data Documentation	25
3.10.3.1 albedo	25
3.10.3.2 emissionPower	25
3.10.3.3 index	25
3.10.3.4 metallic	25
3.10.3.5 roughness	25
3.10.3.6 specular	26
3.11 Ray Struct Reference	26
3.11.1 Detailed Description	26
3.11.2 Member Data Documentation	26
3.11.2.1 direction	26
3.11.2.2 origin	26
3.12 Renderer Class Reference	26
3.12.1 Detailed Description	27
3.12.2 Member Typedef Documentation	27
3.12.2.1 typ_t	27
3.12.3 Constructor & Destructor Documentation	27
3.12.3.1 Renderer()	27
3.12.3.2 ∼Renderer()	28
3.12.4 Member Function Documentation	28
3.12.4.1 Accelerate()	28
3.12.4.2 Accumulate()	28
3.12.4.3 Gamma()	28
3.12.4.4 GetAvailableThreadCount()	28
3.12.4.5 GetFrameIndex()	29
3.12.4.6 GetImage()	29
3.12.4.7 GetUsedThreadCount()	29
3.12.4.8 OnRayMiss()	29

3.12.4.9 OnResize()	29
3.12.4.10 RayDepth()	30
3.12.4.11 Render() [1/2]	30
3.12.4.12 Render() [2/2]	31
3.12.4.13 SaveImage()	31
3.12.4.14 SetUsedThreadCount()	32
3.12.4.15 UsedThreadCount()	32
3.13 Scene Struct Reference	32
3.13.1 Detailed Description	32
3.13.2 Member Function Documentation	33
3.13.2.1 Deserialize()	33
3.13.2.2 Serialize()	33
3.13.3 Member Data Documentation	33
3.13.3.1 boxes	33
3.13.3.2 camera	33
3.13.3.3 materials	33
3.13.3.4 models	34
3.13.3.5 spheres	34
3.13.3.6 triangles	34
3.14 Timer Class Reference	34
3.14.1 Detailed Description	34
3.14.2 Member Function Documentation	34
3.14.2.1 MeasureInMillis()	34
4 File Documentation	35
4.1 AABB.h	
4.2 AccelerationStructure.cpp	
4.3 AccelerationStructure.h	
4.4 Application.cpp	
4.5 Application.h	
4.6 BSDF.cpp	
4.7 BSDF.h	
4.8 BVHNode.h	
4.9 Camera.cpp	
4.10 Camera.h	
4.11 Entrypoint.cpp	
4.12 HitPayload.h	
4.13 Image.cpp	
4.14 Image.h	
	J
4.15 Light.h	
4.15 Light.h	53

Index																		65
4.22 Utilities.hp	p		 															62
4.21 Timer.h .			 															62
4.20 Scene.h			 															59
4.19 Renderer.l	١		 															58
4.18 Renderer.	cpp.		 															54

# **Chapter 1**

# **Class Index**

# 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AABB	5
AccelerationStructure	
Application	
Path tracing application class which holds basic logic	8
BSDF	10
BVHNode	13
Camera	18
HitPayload	21
mage	22
_ight	23
Material	24
Ray	26
Renderer	26
Scene	32
Fimor	24

2 Class Index

# **Chapter 2**

# File Index

# 2.1 File List

Here is a list of all documented files with brief descriptions:

AABB.h	35
AccelerationStructure.cpp	36
AccelerationStructure.h	36
Application.cpp	37
Application.h	45
BSDF.cpp	46
BSDF.h	48
BVHNode.h	48
Camera.cpp	50
Camera.h	51
Entrypoint.cpp	52
HitPayload.h	52
Image.cpp	52
Image.h	53
Light.h	53
Material.h	54
Ray.h	54
Renderer.cpp	54
Renderer.h	58
Scene.h	59
Timer.h	62
Litilities han	62

File Index

# **Chapter 3**

# **Class Documentation**

# 3.1 AABB Struct Reference

#### **Public Member Functions**

- constexpr AABB (const Math::Vector3f &a, const Math::Vector3f &b) noexcept
- constexpr AABB (const AABB &other) noexcept=default
- constexpr AABB (const AABB &a, const AABB &b) noexcept
- constexpr float GetSurfaceArea () const noexcept
- constexpr bool IntersectsRay (const Ray &ray, float t\_min, float t\_max) const noexcept

#### **Static Public Member Functions**

• static constexpr AABB Empty () noexcept

### **Public Attributes**

Math::Vector3f minMath::Vector3f max

### 3.1.1 Detailed Description

Definition at line 6 of file AABB.h.

# 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 AABB() [1/2]

#### 3.1.2.2 AABB() [2/2]

#### 3.1.3 Member Function Documentation

#### 3.1.3.1 Empty()

#### 3.1.3.2 GetSurfaceArea()

# 3.1.3.3 IntersectsRay()

```
00027
00028
               auto one_over_direction = 1.f / ray.direction;
00029
               auto origin = ray.origin;
00030
               auto t0 = (min - origin) * one_over_direction; auto t1 = (max - origin) * one_over_direction;
00031
00032
00033
00034
               if (one_over_direction.x < 0.f) {</pre>
00035
                    std::swap(t0.x, t1.x);
00036
00037
00038
               if (one_over_direction.y < 0.f) {</pre>
00039
                    std::swap(t0.y, t1.y);
00040
               }
00041
00042
               if (one_over_direction.z < 0.f) {</pre>
00043
                   std::swap(t0.z, t1.z);
00044
00045
               t_min = Math::Max(t_min, Math::Max(Math::Max(t0.x, t0.y), t0.z));
00046
00047
               t_max = Math::Min(t_max, Math::Min(Math::Min(t1.x, t1.y), t1.z));
00048
00049
               if (t_max <= t_min) {</pre>
00050
                   return false;
00051
               }
00052
00053
               return true;
00054
           }
```

#### 3.1.4 Member Data Documentation

#### 3.1.4.1 max

```
Math::Vector3f AABB::max
```

Definition at line 7 of file AABB.h.

#### 3.1.4.2 min

```
Math::Vector3f AABB::min
```

Definition at line 7 of file AABB.h.

The documentation for this struct was generated from the following file:

• AABB.h

# 3.2 AccelerationStructure Class Reference

#### **Public Member Functions**

- AccelerationStructure (std::span< HittableObjectPtr > objects) noexcept
- void Update (std::span< HittableObjectPtr > objects) noexcept
- void Hit (const Ray &ray, float tMin, float tMax, HitPayload &payload) const noexcept

# 3.2.1 Detailed Description

Definition at line 13 of file AccelerationStructure.h.

#### 3.2.2 Constructor & Destructor Documentation

#### 3.2.2.1 AccelerationStructure() [1/2]

```
AccelerationStructure::AccelerationStructure ( ) [noexcept]

Definition at line 5 of file AccelerationStructure.cpp.

00005
00006 m_Root = new BVHNode(new NonHittable());
```

#### 3.2.2.2 AccelerationStructure() [2/2]

```
AccelerationStructure::AccelerationStructure ( std::span< HittableObjectPtr > objects ) [noexcept]
```

#### Definition at line 9 of file AccelerationStructure.cpp.

00007 }

#### 3.2.2.3 ∼AccelerationStructure()

```
AccelerationStructure::~AccelerationStructure ( ) [noexcept]
```

```
Definition at line 13 of file AccelerationStructure.cpp.

00013 {
00014 if (m_Root != nullptr) {
00015 BVHNode::FreeMemory(m_Root);
00016 }
00017 }
```

#### 3.2.3 Member Function Documentation

#### 3.2.3.1 Hit()

Definition at line 31 of file AccelerationStructure.cpp.

#### 3.2.3.2 Update()

Definition at line 19 of file AccelerationStructure.cpp.

```
00020
          if (m_Root != nullptr) {
00021
              BVHNode::FreeMemory(m_Root);
00022
         }
00023
00024
         if (objects.empty()) {
00025
             m_Root = new BVHNode(new NonHittable());
00026
             m_Root = BVHNode::MakeHierarchySAH(objects, 0, static_cast<int>(objects.size()));
00027
         }
00028
00029 }
```

The documentation for this class was generated from the following files:

- · AccelerationStructure.h
- · AccelerationStructure.cpp

# 3.3 Application Class Reference

Path tracing application class which holds basic logic.

```
#include <Application.h>
```

#### **Public Member Functions**

- · Application (int windowWidth, int windowHeight) noexcept
- int Run () noexcept

# 3.3.1 Detailed Description

Path tracing application class which holds basic logic.

Definition at line 15 of file Application.h.

#### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 Application()

#### Definition at line 21 of file Application.cpp.

```
00021
00022
                                 m_InitialWindowWidth(windowWidth), m_InitialWindowHeight(windowHeight),
                                 m_LastViewportWidth(-1), m_LastViewportHeight(-1),
m_Renderer(windowWidth, windowHeight),
00023
00024
00025
                                 m_TotalRenderTime(0.f), m_LastRenderTime(0.f) {
00026
00027
                                 m_SaveImageFilePath = new char[c_AnyInputFilePathLength];
                                m_SceneFilePath = new char[c_AnyInputFilePathLength];
m_ModelFilePath = new char[c_AnyInputFilePathLength];
00028
00029
00030
                                 m_MaterialDirectory = new char[c_AnyInputFilePathLength];
00031
                                 memset(m_SaveImageFilePath, 0, sizeof(m_SaveImageFilePath));
00032
                                 memset(m_SceneFilePath, 0, sizeof(m_SceneFilePath));
00033
                                 {\tt memset} \; ({\tt m\_ModelFilePath}, \;\; {\tt 0, \;\; sizeof} \; ({\tt m\_ModelFilePath}) \, ) \; ;
00034
                                 memset(m_MaterialDirectory, 0, sizeof(m_MaterialDirectory));
00035
00036
                                 m_AddMaterial.albedo = {0.f, 0.f, 0.f};
00037
                                 m_AddMaterial.metallic = 0.f;
00038
                                 m_AddMaterial.specular = 0.f;
00039
                                 m_AddMaterial.roughness = 0.f;
00040
                                 m\_AddMaterial.emissionPower = 0.f;
00041
                                 m AddMaterial.index = -1;
00042
                                 m_AddSphere = Shapes::Sphere(Math::Vector3f(0.f), 0.f, nullptr);
00043
00044
                                  \texttt{m\_AddTriangle} = \texttt{Shapes::} \texttt{Triangle} (\texttt{Math::} \texttt{Vector3f} (\texttt{0.f}) \text{, } \texttt{Math::} \texttt{Vector3f} (\texttt{0.f}) \text{, } \texttt
                   nullptr);
00045
                                 m_AddBox = Shapes::Box(Math::Vector3f(0.f), Math::Vector3f(0.f), nullptr);
00046
00047
                                 m AddSphereMaterialIndex = -1;
00048
                                 m_AddTriangleMaterialIndex = -1;
00049
                                 m_AddBoxMaterialIndex = -1;
00050
00051
                                 m_RayMissColor = Math::Vector3f(0.f);
00052
00053
                                 m Scene.camera = Camera(windowWidth, windowHeight);
00054
00055
                                  LoadSceneFromFile(c_DefaultScenePath);
00056 }
```

#### 3.3.3 Member Function Documentation

#### 3.3.3.1 Run()

```
int Application::Run ( ) [noexcept]
Definition at line 58 of file Application.cpp.
          if (!glfwInit()) {
00060
              std::cerr « "Failed to initialize GLFW\n";
00061
              return -1;
00062
00063
00064 #if defined(IMGUI_IMPL_OPENGL_ES2)
          const char* glsl_version = "#version 100";
00066
          glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 2);
00067
          glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 0);
00068
          glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_ES_API);
00069 #elif defined(_APPLE__)
00070 const char* glsl_version = "#version 150";
00071
          glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
00072
          glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 2);
00073
          glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
00074
          glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
00075 #else
         const char* glsl_version = "#version 130";
00076
          glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
00077
00078
          glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 0);
00079 #endif
08000
00081
          m_Window = glfwCreateWindow(m_InitialWindowWidth, m_InitialWindowHeight, c_WindowTitle, nullptr,
     nullptr);
00082
         if (m_Window == nullptr) {
00083
              std::cerr « "Failed to create window\n";
00084
              return -1;
00085
          }
00086
          GLFWimage *icon = new GLFWimage();
00087
00088
          icon->pixels = stbi_load("icon/icon.png", &icon->width, &icon->height, 0, 4);
00089
          glfwSetWindowIcon(m_Window, 1, icon);
00090
          stbi_image_free(icon->pixels);
00091
          delete icon;
00092
00093
          glfwMakeContextCurrent(m_Window);
00094
00095
          ImGui::CreateContext();
00096
          ImGuiIO &io = ImGui::GetIO();
00097
          io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;
00098
00099
          UpdateThemeStyle();
00100
00101
          ImGui_ImplGlfw_InitForOpenGL(m_Window, true);
00102
          ImGui_ImplOpenGL3_Init(glsl_version);
00103
          MainLoop();
00104
00105
          ImGui_ImplOpenGL3_Shutdown();
00106
00107
          ImGui_ImplGlfw_Shutdown();
00108
          ImGui::DestroyContext();
00109
00110
          glfwDestroyWindow(m_Window);
00111
          glfwTerminate();
00112
00113
          return 0;
```

The documentation for this class was generated from the following files:

- · Application.h
- · Application.cpp

# 3.4 BSDF Class Reference

#### **Public Member Functions**

- constexpr BSDF (const Material \*material) noexcept
- Math::Vector3f Sample (const Ray &ray, const HitPayload &payload, Math::Vector3f &throughput) noexcept

3.4 BSDF Class Reference 11

# 3.4.1 Detailed Description

Definition at line 9 of file BSDF.h.

#### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 BSDF()

#### 3.4.3 Member Function Documentation

#### 3.4.3.1 Sample()

Math::Vector3f BSDF::Sample (

```
const Ray & ray,
                const HitPayload & payload,
                Math::Vector3f & throughput ) [noexcept]
Definition at line 4 of file BSDF.cpp.
00004
          float diffuseRatio = 0.5f * (1.f - m_Material->metallic);
float specularRatio = 1.f - diffuseRatio;
00005
00006
80000
           Math:: Vector3f V = -ray.direction;
00009
00010
          Math:: Vector3f reflectionDirection;
          if (Utilities::RandomFloatInZeroToOne() < diffuseRatio) {</pre>
00011
00012
               reflectionDirection = Utilities::RandomInHemisphere(payload.normal);
00013
00014
               Math::Vector3f halfVec;
00015
               {
00016
                    Math:: Vector2f Xi{Utilities:: RandomFloatInZeroToOne(),
     Utilities::RandomFloatInZeroToOne() };
00017
                   Math:: Vector3f N = payload.normal;
00018
00019
                   float a = m_Material->roughness * m_Material->roughness;
00020
                   float phi = Math::Constants::Tau<float> * Xi.x;
float cosTheta = Math::Sqrt((1.f - Xi.y) / (1.f + (a * a - 1.f) * Xi.y));
float sinTheta = Math::Sqrt(1.f - cosTheta * cosTheta);
00021
00022
00023
00024
00025
                    Math::Vector3f H;
                    H.x = Math::Cos(phi) * sinTheta;
H.y = Math::Sin(phi) * sinTheta;
00026
00027
                   H.z = cosTheta;
00028
00029
                   Math::Vector3f up = Math::Abs(N.z) < 0.999f ? Math::Vector3f(0.0, 0.0, 1.0) :
00030
     Math:: Vector3f(1.0, 0.0, 0.0);
00031
                    Math::Vector3f tangent = Math::Normalize(Math::Cross(up, N));
00032
                    Math::Vector3f bitangent = Math::Cross(N, tangent);
00033
00034
                    halfVec = tangent * H.x + bitangent * H.v + N * H.z;
                   halfVec = Math::Normalize(halfVec);
00035
00036
               }
00037
00038
               reflectionDirection = Math::Normalize(2.f * Math::Dot(V, halfVec) * halfVec - V);
00039
          }
00040
00041
          auto DistributionGGX = [](const Math::Vector3f &N, const Math::Vector3f &H, float roughness) {
00042
              float a = roughness * roughness;
               float a2 = a * a;
```

```
float NdotH = Math::Max(Math::Dot(N, H), 0.f);
               float NdotH2 = NdotH * NdotH;
00045
00046
00047
               float nom = a2;
               float denom = (NdotH2 * (a2 - 1.f) + 1.f);
00048
00049
               denom = Math::Constants::Pi<float> * denom * denom;
00050
00051
               return nom / denom;
00052
          };
00053
          auto GeometrySchlickGGX = [](float NdotV, float roughness) {
00054
               float r = (roughness + 1.f);
float k = (r * r) / 8.f;
00055
00056
00057
00058
               float nom = NdotV;
00059
               float denom = NdotV * (1.f - k) + k;
00060
00061
               return nom / denom;
00062
00063
           auto GeometrySmith = [&](const Math::Vector3f &N, const Math::Vector3f &V, const Math::Vector3f
00064
      &L, float roughness) {
00065
               float NdotV = Math::Abs(Math::Dot(N, V));
               float NdotL = Math::Abs(Math::Dot(N, L));
00066
00067
               float ggx2 = GeometrySchlickGGX(NdotV, roughness);
               float ggx1 = GeometrySchlickGGX(NdotL, roughness);
00068
00069
00070
               return qqx1 * qqx2;
00071
          };
00072
           auto FresnelSchlick = [](float cosTheta, const Math::Vector3f &F0) {
00073
00074
              return F0 + (1.f - F0) * Math::Pow(1.f - cosTheta, 5.f);
00075
00076
00077
           auto SpecularBRDF = [](float D, float G, const Math::Vector3f &F, const Math::Vector3f &V, const
     Math::Vector3f &L, const Math::Vector3f &N) {
    float NdotL = Math::Abs(Math::Dot(N, L));
00078
               float NdotV = Math::Abs(Math::Dot(N, V));
00079
00080
               Math::Vector3f nominator = D * G * F;
float denominator = 4.f * NdotV * NdotL + 0.001f;
00081
00082
00083
               Math::Vector3f specularBrdf = nominator / denominator;
00084
00085
              return specularBrdf;
00086
          };
00087
00088
           auto DiffuseBRDF = [](const Math::Vector3f &albedo) {
00089
               return albedo * Math::Constants::InversePi<float>;
00090
00091
          auto ImportanceSampleGGXPDF = [](float NDF, float NdotH, float VdotH) {
    return NDF * NdotH / (4.f * VdotH);
00092
00093
00094
00095
           auto CosineSamplingPDF = [](float NdotL) {
00096
00097
              return NdotL * Math::Constants::InversePi<float>;
00098
00099
          Math::Vector3f L = reflectionDirection;
Math::Vector3f H = Math::Normalize(V + L);
00100
00101
00102
00103
           float NdotL = Math::Abs(Math::Dot(payload.normal, L));
00104
           float NdotH = Math::Abs(Math::Dot(payload.normal, H));
00105
           float VdotH = Math::Abs(Math::Dot(V, H));
00106
00107
           float NdotV = Math::Abs(Math::Dot(payload.normal, V));
00108
           Math:: Vector3f F0 = Math:: Vector3f(0.08f, 0.08f, 0.08f);
00109
00110
          F0 = Math::Lerp(F0 * m_Material->specular, m_Material->albedo, m_Material->metallic);
00111
00112
           float NDF = DistributionGGX(payload.normal, H, m_Material->roughness);
00113
           float G = GeometrySmith(payload.normal, V, L, m_Material->roughness);
00114
          Math::Vector3f F = FresnelSchlick(Math::Max(Math::Dot(H, V), 0.f), F0);
00115
00116
          Math::Vector3f kS = F;
Math::Vector3f kD = 1.f - kS;
00117
00118
           kD *= 1.0 - m_Material->metallic;
00119
           Math::Vector3f specularBrdf = SpecularBRDF(NDF, G, F, V, L, payload.normal);
00120
00121
00122
           float specularPdf = ImportanceSampleGGXPDF(NDF, NdotH, VdotH);
00123
00124
           Math::Vector3f diffuseBrdf = DiffuseBRDF(m_Material->albedo);
00125
           float diffusePdf = CosineSamplingPDF(NdotL);
00126
           Math::Vector3f totalBrdf = (diffuseBrdf * kD + specularBrdf) * NdotL;
00127
00128
           float totalPdf = diffuseRatio * diffusePdf + specularRatio * specularPdf;
```

The documentation for this class was generated from the following files:

- BSDF.h
- BSDF.cpp

# 3.5 BVHNode Struct Reference

#### **Public Member Functions**

- constexpr BVHNode (BVHNode \*left, BVHNode \*right) noexcept
- constexpr BVHNode (const HittableObject \*object) noexcept
- · constexpr bool IsTerminating () const noexcept
- · bool Hit (const Ray &ray, float tMin, float tMax, HitPayload &payload) noexcept

#### Static Public Member Functions

- static BVHNode \* MakeHierarchySAH (std::span< HittableObjectPtr > objects, int low, int high) noexcept
- static BVHNode \* MakeHierarchyNaive (std::span< HittableObjectPtr > objects, int low, int high) noexcept
- static constexpr void FreeMemory (BVHNode \*node) noexcept

#### **Public Attributes**

- AABB aabb = AABB::Empty()
- BVHNode \* left = nullptr
- BVHNode \* right = nullptr
- const HittableObject \* object = nullptr

#### **Static Public Attributes**

- static const std::function < bool(HittableObjectPtr, HittableObjectPtr) > c\_ComparatorsSAH [3]
- static const std::function< bool(HittableObjectPtr, HittableObjectPtr)> c\_ComparatorsNaive [3]

# 3.5.1 Detailed Description

Definition at line 13 of file BVHNode.h.

#### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 BVHNode() [1/2]

#### 3.5.3 Member Function Documentation

#### 3.5.3.1 FreeMemory()

```
static constexpr void BVHNode::FreeMemory (
              BVHNode * node ) [inline], [static], [constexpr], [noexcept]
Definition at line 154 of file BVHNode.h.
00154
                                                                 {
             if (node == nullptr) {
00155
00156
                 return;
00157
00158
00159
             FreeMemory(node->left);
00160
             FreeMemory(node->right);
00161
00162
             delete node;
00163
```

#### 3.5.3.2 Hit()

#### Definition at line 30 of file BVHNode.h.

```
00030
00031
              if (!aabb.IntersectsRay(ray, tMin, tMax)) {
00032
                 return false;
              }
00033
00034
00035
              if (IsTerminating()) {
00036
                  return object->Hit(ray, tMin, tMax, payload);
00037
00038
00039
             bool anyHit = left->Hit(ray, tMin, tMax, payload);
00040
              anyHit |= right->Hit(ray, tMin, Math::Min(tMax, payload.t), payload);
00041
00042
              return anyHit;
00043
         }
```

#### 3.5.3.3 IsTerminating()

#### 3.5.3.4 MakeHierarchyNaive()

#### Definition at line 108 of file BVHNode.h.

```
00108
00109
              AABB aabb = AABB::Empty();
00110
              for (int i = low; i < high; ++i) {</pre>
00111
                 aabb = AABB(aabb, objects[i]->GetBoundingBox());
00112
00113
00114
              int longestAxisIndex = 0:
00115
              float longestAxisLength = aabb.max.x - aabb.min.x;
00116
00117
              if (aabb.max.y - aabb.min.y > longestAxisLength) {
00118
                  longestAxisIndex = 1;
                  longestAxisLength = aabb.max.y - aabb.min.y;
00119
00120
              }
00121
              if (aabb.max.z - aabb.min.z > longestAxisLength) {
00123
                  longestAxisIndex = 2;
00124
                  longestAxisLength = aabb.max.z - aabb.min.z;
00125
              }
00126
00127
              auto comparator = c_ComparatorsNaive[longestAxisIndex];
00128
00129
              if (low + 1 == high) {
00130
                  return new BVHNode(objects[low]);
00131
00132
00133
              std::sort(objects.begin() + low, objects.begin() + high, comparator);
00134
00135
              int mid = (low + high) / 2;
              BVHNode *left = MakeHierarchyNaive(objects, low, mid);
00136
00137
              BVHNode *right = MakeHierarchyNaive(objects, mid, high);
00138
00139
              return new BVHNode(left, right);
00140
          }
```

#### 3.5.3.5 MakeHierarchySAH()

#### Definition at line 45 of file BVHNode.h.

```
00045

00046
    if (low + 1 == high) {
        return new BVHNode(objects[low]);

00048
    }

00049

00050    int n = high - low;

00051    std::vector<AABB> pref(n + 1);

00052    std::vector<AABB> suff(n + 1);
```

```
float minValue = Math::Constants::Infinity<float>;
               int mid = -1;
00055
00056
               int dimension = -1;
00057
               for (int d = 0; d < 3; ++d) {
00058
00059
                   std::sort(objects.begin() + low, objects.begin() + high, c_ComparatorsSAH[d]);
00061
                   pref[0] = AABB::Empty();
                   for (int i = 0; i < n; ++i) {
    pref[i + 1] = AABB(pref[i], objects[i + low]->GetBoundingBox());
00062
00063
00064
00065
                   suff[n] = AABB::Empty();
for (int i = n - 1; i >= 0; --i) {
00066
00067
00068
                       suff[i] = AABB(objects[i + low]->GetBoundingBox(), suff[i + 1]);
00069
00070
00071
                   float minValueAlongAxis = Math::Constants::Infinity<float>;
00072
                   int index = -1;
00073
                   for (int i = 0; i < n; ++i) {
     float value = pref[i + 1].GetSurfaceArea() * (float)(i + 1) + suff[i + 1].GetSurfaceArea() * (float)(n - i - 1);
00074
00075
                       if (value < minValueAlongAxis) {</pre>
00076
                            minValueAlongAxis = value;
00077
                            index = i + low;
00078
00079
                   }
08000
00081
                   if (minValueAlongAxis < minValue) {</pre>
00082
                       minValue = minValueAlongAxis;
00083
                       mid = index + 1;
00084
                       dimension = d;
00085
00086
               }
00087
00088
               std::sort(objects.begin() + low, objects.begin() + high, c_ComparatorsSAH[dimension]);
00089
00090
               BVHNode *left = MakeHierarchySAH(objects, low, mid);
00091
               BVHNode *right = MakeHierarchySAH(objects, mid, high);
00092
00093
               return new BVHNode(left, right);
00094
          }
```

#### 3.5.4 Member Data Documentation

#### 3.5.4.1 aabb

```
AABB BVHNode::aabb = AABB::Empty()
```

Definition at line 14 of file BVHNode.h.

# 3.5.4.2 c ComparatorsNaive

```
const std::function<bool(HittableObjectPtr, HittableObjectPtr)> BVHNode::c_ComparatorsNaive[3]
[inline], [static]
```

```
Initial value:
```

```
= {
        [](HittableObjectPtr a, HittableObjectPtr b) {
            return a->GetBoundingBox().min.x < b->GetBoundingBox().min.x;
        },
        [](HittableObjectPtr a, HittableObjectPtr b) {
            return a->GetBoundingBox().min.y < b->GetBoundingBox().min.y;
        },
        [](HittableObjectPtr a, HittableObjectPtr b) {
            return a->GetBoundingBox().min.z < b->GetBoundingBox().min.z;
        }
    }
}
```

Definition at line 142 of file BVHNode.h.

```
00142
```

#### 3.5.4.3 c\_ComparatorsSAH

const std::function<bool(HittableObjectPtr, HittableObjectPtr)> BVHNode::c\_ComparatorsSAH[3]
[inline], [static]

#### Initial value:

```
[](HittableObjectPtr a, HittableObjectPtr b){
    return a->GetCentroid().x < b->GetCentroid().x;
},
[](HittableObjectPtr a, HittableObjectPtr b){
    return a->GetCentroid().y < b->GetCentroid().y;
},
[](HittableObjectPtr a, HittableObjectPtr b){
    return a->GetCentroid().z < b->GetCentroid().z;
}
```

#### Definition at line 96 of file BVHNode.h.

```
00097
              [](HittableObjectPtr a, HittableObjectPtr b){
00098
                 return a->GetCentroid().x < b->GetCentroid().x;
00099
              [](HittableObjectPtr a, HittableObjectPtr b){
00100
00101
                 return a->GetCentroid().y < b->GetCentroid().y;
00102
00103
              [](HittableObjectPtr a, HittableObjectPtr b){
00104
                 return a->GetCentroid().z < b->GetCentroid().z;
             }
00105
00106
         };
```

#### 3.5.4.4 left

```
BVHNode* BVHNode::left = nullptr
```

Definition at line 15 of file BVHNode.h.

# 3.5.4.5 object

```
const HittableObject* BVHNode::object = nullptr
```

Definition at line 16 of file BVHNode.h.

#### 3.5.4.6 right

```
BVHNode * BVHNode::right = nullptr
```

Definition at line 15 of file BVHNode.h.

The documentation for this struct was generated from the following file:

• BVHNode.h

#### 3.6 Camera Class Reference

#### **Public Member Functions**

- · void OnViewportResize (int viewportWidth, int viewportHeight) noexcept
- void ComputeRayDirections () noexcept
- constexpr std::span< const Math::Vector3f > GetRayDirections () const noexcept
- · constexpr Math::Vector3f GetPosition () const noexcept
- constexpr Math::Vector3f & Position () noexcept
- constexpr Math::Vector3f GetTarget () const noexcept
- constexpr Math::Vector3f & Target () noexcept
- constexpr Math::Vector3f GetUp () const noexcept
- constexpr Math::Vector3f & Up () noexcept
- constexpr float GetVerticalFovInDegrees () const noexcept
- · constexpr float & VerticalFovInDegrees () noexcept

# 3.6.1 Detailed Description

Definition at line 9 of file Camera.h.

#### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 Camera()

#### Definition at line 4 of file Camera.cpp.

#### 3.6.3 Member Function Documentation

#### 3.6.3.1 ComputeRayDirections()

```
void Camera::ComputeRayDirections ( ) [noexcept]
Definition at line 21 of file Camera.cpp.
00021
00022
           float verticalFovInRadians = Math::ToRadians(m_VerticalFovInDegrees);
00023
00024
           Math::Vector3f forward = m_Target - m_Position;
00025
           float focalLength = Math::Length(forward);
           float viewportWorldHeight = 2.f * Math::Tan(verticalFovInRadians * 0.5f) * focalLength; float viewportWorldWidth = (viewportWorldHeight * m_ViewportWidth) / m_ViewportHeight;
00026
00027
00028
00029
           Math::Vector3f w = Math::Normalize(forward);
00030
           Math::Vector3f u = Math::Normalize(Math::Cross(w, m_Up));
00031
           Math::Vector3f v = Math::Normalize(Math::Cross(w, u));
00032
           \label{eq:math::Vector3f} \begin{tabular}{ll} Math::Vector3f borizontal = u * viewportWorldHeight; \\ Math::Vector3f borizontal = v * viewportWorldHeight; \\ \end{tabular}
00033
00034
00035
           Math::Vector3f leftUpper = forward - horizontal * 0.5f - vertical * 0.5f;
00036
00037
           for (int i = 0; i < m_ViewportHeight; ++i) {</pre>
00038
             for (int j = 0; j < m_ViewportWidth; ++j) {</pre>
                    float uScale = (float)(j + Utilities::RandomFloatInNegativeHalfToHalf()) /
00039
      (m_ViewportWidth - 1);
00040
                    float vScale = (float)(i + Utilities::RandomFloatInNegativeHalfToHalf()) /
       (m_ViewportHeight - 1);
                   m_RayDirections[m_ViewportWidth * i + j] = Math::Normalize(leftUpper + horizontal * uScale
      + vertical * vScale);
00042
              }
00043
00044 }
3.6.3.2 GetPosition()
constexpr Math::Vector3f Camera::GetPosition ( ) const [inline], [constexpr], [noexcept]
Definition at line 25 of file Camera.h.
00025
00026
               return m_Position;
00027
3.6.3.3 GetRayDirections()
constexpr std::span< const Math::Vector3f > Camera::GetRayDirections ( ) const [inline],
[constexpr], [noexcept]
Definition at line 21 of file Camera.h.
00021
00022
               return m_RayDirections;
00023
           }
3.6.3.4 GetTarget()
constexpr Math::Vector3f Camera::GetTarget ( ) const [inline], [constexpr], [noexcept]
Definition at line 33 of file Camera.h.
00033
00034
               return m_Target;
00035
```

#### 3.6.3.5 GetUp()

```
constexpr Math::Vector3f Camera::GetUp ( ) const [inline], [constexpr], [noexcept]
Definition at line 41 of file Camera.h.
00042
             return m_Up;
00043
3.6.3.6 GetVerticalFovInDegrees()
constexpr float Camera::GetVerticalFovInDegrees ( ) const [inline], [constexpr], [noexcept]
Definition at line 49 of file Camera.h.
                                                                 {
00050
             return m_VerticalFovInDegrees;
00051
3.6.3.7 OnViewportResize()
void Camera::OnViewportResize (
             int viewportWidth,
              int viewportHeight ) [noexcept]
Definition at line 10 of file Camera.cpp.
00010
00011
         if (m_ViewportWidth == viewportWidth && m_ViewportHeight == viewportHeight) {
00012
             return;
00013
         }
00014
00015
         m_ViewportWidth = viewportWidth;
00016
         m_ViewportHeight = viewportHeight;
00017
00018
         m_RayDirections.resize(m_ViewportWidth * m_ViewportHeight);
00019 }
3.6.3.8 Position()
constexpr Math::Vector3f & Camera::Position ( ) [inline], [constexpr], [noexcept]
Definition at line 29 of file Camera.h.
                                                     {
00029
00030
             return m_Position;
00031
3.6.3.9 Target()
constexpr Math::Vector3f & Camera::Target ( ) [inline], [constexpr], [noexcept]
Definition at line 37 of file Camera.h.
                                                   {
00037
00038
             return m_Target;
00039
```

#### 3.6.3.10 Up()

#### 3.6.3.11 VerticalFovInDegrees()

The documentation for this class was generated from the following files:

- · Camera.h
- · Camera.cpp

# 3.7 HitPayload Struct Reference

#### **Public Attributes**

- float t
- Math::Vector3f normal
- const Material \* material

# 3.7.1 Detailed Description

Definition at line 7 of file HitPayload.h.

#### 3.7.2 Member Data Documentation

#### 3.7.2.1 material

```
const Material* HitPayload::material
```

Definition at line 10 of file HitPayload.h.

#### 3.7.2.2 normal

Math::Vector3f HitPayload::normal

Definition at line 9 of file HitPayload.h.

#### 3.7.2.3 t

```
float HitPayload::t
```

Definition at line 8 of file HitPayload.h.

The documentation for this struct was generated from the following file:

· HitPayload.h

# 3.8 Image Class Reference

#### **Public Member Functions**

- Image (int width, int height, const uint32\_t \*data) noexcept
- void UpdateData (const uint32 t \*data) noexcept
- constexpr unsigned int GetDescriptor () const noexcept

# 3.8.1 Detailed Description

Definition at line 7 of file Image.h.

#### 3.8.2 Constructor & Destructor Documentation

#### 3.8.2.1 Image()

```
Image::Image (
          int width,
          int height,
          const uint32_t * data ) [noexcept]
```

# Definition at line 4 of file Image.cpp.

```
00004
00005
             m_Width(width), m_Height(height) {
            glGenTextures(1, &m_Descriptor);
glBindTexture(GL_TEXTURE_2D, m_Descriptor);
00006
00007
00008
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00009
00010
00011
             glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, m_Width, m_Height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
00012
       (void*)data);
00013
00014
             glBindTexture(GL_TEXTURE_2D, 0);
00015 }
```

#### 3.8.2.2 ∼Image()

```
{\tt Image::}{\sim}{\tt Image ( ) } {\tt [noexcept]}
```

# Definition at line 17 of file Image.cpp.

```
00017 {
00018 glDeleteTextures(1, &m_Descriptor);
00019 }
```

#### 3.8.3 Member Function Documentation

#### 3.8.3.1 GetDescriptor()

```
constexpr unsigned int Image::GetDescriptor ( ) const [inline], [constexpr], [noexcept]
Definition at line 17 of file Image.h.
00018
               return m_Descriptor;
00019
3.8.3.2 UpdateData()
void Image::UpdateData (
                const uint32_t * data ) [noexcept]
Definition at line 21 of file Image.cpp.
           glBindTexture(GL_TEXTURE_2D, m_Descriptor);
           glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00024
00025
00026
           glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, m_Width, m_Height, GL_RGBA, GL_UNSIGNED_BYTE,
00027
      (void*)data);
00028
00029
           glBindTexture(GL_TEXTURE_2D, 0);
00030 }
```

The documentation for this class was generated from the following files:

- · Image.h
- · Image.cpp

# 3.9 Light Class Reference

#### **Public Member Functions**

- · constexpr Light (const HittableObjectPtr object, const Math::Vector3f &emission) noexcept
- const HittableObject \* GetObject () const noexcept
- Math::Vector3f Sample (const Ray &lightRay, const HitPayload &objectHitPayload, const HitPayload &light← HitPayload, float distance, float distanceSquared) const noexcept

# 3.9.1 Detailed Description

Definition at line 9 of file Light.h.

#### 3.9.2 Constructor & Destructor Documentation

#### 3.9.2.1 Light()

# 3.9.3 Member Function Documentation

#### 3.9.3.1 GetObject()

```
const HittableObject * Light::GetObject ( ) const [inline], [noexcept]
Definition at line 14 of file Light.h.
00015
              return m_Object;
00016
3.9.3.2 Sample()
Math::Vector3f Light::Sample (
              const Ray & lightRay,
              const HitPayload & objectHitPayload,
              const HitPayload & lightHitPayload,
              float distance,
              float distanceSquared ) const [inline], [noexcept]
Definition at line 18 of file Light.h.
00018
00019
              constexpr float distanceEpsilon = 0.01f;
00020
             if (Math::Abs(lightHitPayload.t - distance) > distanceEpsilon) {
                 return Math::Vector3f(0.f);
00022
00023
00024
m_Object->GetArea());
00025
             float pdf = distanceSquared / (Math::Dot(lightHitPayload.normal, -lightRay.direction) *
00026
             constexpr float pdfEpsilon = 0.01f;
00027
             if (pdf <= pdfEpsilon) {</pre>
00028
                 return Math::Vector3f(0.f);
00029
00030
             Math::Vector3f brdf = objectHitPavload.material->albedo * Math::Constants::InversePi<float> *
00031
     Math::Dot(objectHitPayload.normal, lightRay.direction) * m_Emission;
00032
00033
              return brdf / pdf;
00034
```

The documentation for this class was generated from the following file:

· Light.h

# 3.10 Material Struct Reference

#### **Public Member Functions**

constexpr Math::Vector3f GetEmission () const noexcept

#### **Public Attributes**

- Math::Vector3f albedo
- · float metallic
- · float specular
- float roughness
- float emissionPower
- int index

# 3.10.1 Detailed Description

Definition at line 6 of file Material.h.

#### 3.10.2 Member Function Documentation

#### 3.10.2.1 GetEmission()

# 3.10.3 Member Data Documentation

#### 3.10.3.1 albedo

Math::Vector3f Material::albedo

Definition at line 7 of file Material.h.

#### 3.10.3.2 emissionPower

float Material::emissionPower

Definition at line 11 of file Material.h.

#### 3.10.3.3 index

int Material::index

Definition at line 13 of file Material.h.

### 3.10.3.4 metallic

float Material::metallic

Definition at line 8 of file Material.h.

# **3.10.3.5 roughness**

float Material::roughness

Definition at line 10 of file Material.h.

#### 3.10.3.6 specular

```
float Material::specular
```

Definition at line 9 of file Material.h.

The documentation for this struct was generated from the following file:

· Material.h

# 3.11 Ray Struct Reference

#### **Public Attributes**

Math::Vector3f originMath::Vector3f direction

# 3.11.1 Detailed Description

Definition at line 6 of file Ray.h.

#### 3.11.2 Member Data Documentation

#### 3.11.2.1 direction

Math::Vector3f Ray::direction

Definition at line 7 of file Ray.h.

#### 3.11.2.2 origin

Math::Vector3f Ray::origin

Definition at line 7 of file Ray.h.

The documentation for this struct was generated from the following file:

• Ray.h

# 3.12 Renderer Class Reference

#### **Public Types**

using typ\_t = HittableObject\*

#### **Public Member Functions**

- · Renderer (int width, int height) noexcept
- void Render (const Camera &camera, std::span< const HittableObjectPtr > objects, std::span< const Light</li>
   lightSources, std::span< const Material > materials) noexcept
- void Render (const Camera &camera, const AccelerationStructure &accelerationStructure, std::span< const Light > lightSources, std::span< const Material > materials) noexcept
- · constexpr bool & Accumulate () noexcept
- · constexpr bool & Accelerate () noexcept
- constexpr int GetFrameIndex () const noexcept
- constexpr Image \* GetImage () const noexcept
- void SaveImage (const char \*filename) const noexcept
- · void OnResize (int width, int height) noexcept
- constexpr int GetAvailableThreadCount () const noexcept
- constexpr int GetUsedThreadCount () const noexcept
- constexpr void SetUsedThreadCount (int usedThreads) noexcept
- constexpr int & UsedThreadCount () noexcept
- void OnRayMiss (std::function < Math::Vector3f(const Ray &) > onRayMiss) noexcept
- · constexpr int & RayDepth () noexcept
- · constexpr float & Gamma () noexcept

### 3.12.1 Detailed Description

Definition at line 17 of file Renderer.h.

# 3.12.2 Member Typedef Documentation

# 3.12.2.1 typ\_t

```
using Renderer::typ_t = HittableObject*
```

Definition at line 25 of file Renderer.h.

#### 3.12.3 Constructor & Destructor Documentation

#### 3.12.3.1 Renderer()

```
Renderer::Renderer (
          int width,
          int height ) [noexcept]
```

#### Definition at line 12 of file Renderer.cpp.

```
00012
00013
           m_Width(width), m_Height(height) {
00014
           m_ImageData = new uint32_t[m_Width * m_Height];
           m_Image = new Image(m_Width, m_Height, m_ImageData);
m_AccumulationData = new Math::Vector4f[m_Width * m_Height];
00015
00016
00017
00018
           m_AvailableThreads = std::thread::hardware_concurrency();
00019
           m UsedThreads = 1:
00020
           m_LinesPerThread = (m_Height + m_UsedThreads - 1) / m_UsedThreads;
00021 }
```

#### 3.12.3.2 ∼Renderer()

```
Renderer::\simRenderer ( ) [noexcept]
```

#### Definition at line 23 of file Renderer.cpp.

```
00024
          if (m_Image != nullptr) {
00025
             delete m_Image;
00026
00027
         if (m_ImageData != nullptr) {
00028
             delete[] m_ImageData;
00029
00030
         if (m_AccumulationData != nullptr) {
00031
             delete[] m_AccumulationData;
         }
00032
00033 }
```

#### 3.12.4 Member Function Documentation

#### 3.12.4.1 Accelerate()

#### 3.12.4.2 Accumulate()

```
constexpr bool & Renderer::Accumulate ( ) [inline], [constexpr], [noexcept]

Definition at line 31 of file Renderer.h.
```

```
00032 return m_Accumulate;
00033 }
```

#### 3.12.4.3 Gamma()

00053

#### 3.12.4.4 GetAvailableThreadCount()

#### 3.12.4.5 GetFrameIndex()

```
constexpr int Renderer::GetFrameIndex ( ) const [inline], [constexpr], [noexcept]
Definition at line 39 of file Renderer.h.
00039
                                                      {
00040
             return m_FrameIndex;
00041
3.12.4.6 GetImage()
constexpr Image * Renderer::GetImage ( ) const [inline], [constexpr], [noexcept]
Definition at line 43 of file Renderer.h.
00043
                                                    {
00044
             return m_Image;
00045
3.12.4.7 GetUsedThreadCount()
constexpr int Renderer::GetUsedThreadCount ( ) const [inline], [constexpr], [noexcept]
Definition at line 55 of file Renderer.h.
                                                           {
00055
00056
             return m_UsedThreads;
00057
3.12.4.8 OnRayMiss()
void Renderer::OnRayMiss (
              std::function< Math::Vector3f(const Ray &) > onRayMiss ) [inline], [noexcept]
Definition at line 68 of file Renderer.h.
00068
                                                                                        {
00069
             m_OnRayMiss = onRayMiss;
00070
3.12.4.9 OnResize()
void Renderer::OnResize (
              int width,
              int height ) [noexcept]
Definition at line 39 of file Renderer.cpp.
00039
00040
          if (m_Width == width && m_Height == height) {
00041
             return;
00042
         }
00043
00044
         m_Width = width;
00045
         m_Height = height;
00046
00047
          if (m_ImageData != nullptr) {
00048
             delete m_ImageData;
             m_ImageData = new uint32_t[m_Width * m_Height];
00049
00050
00051
         if (m_Image != nullptr) {
00052
             delete m_Image;
00053
             m_Image = new Image(m_Width, m_Height, m_ImageData);
00054
00055
          if (m_AccumulationData != nullptr) {
00056
             delete m_AccumulationData;
00057
             m_AccumulationData = new Math::Vector4f[m_Width * m_Height];
00058
         }
00059 }
```

30 Class Documentation

#### 3.12.4.10 RayDepth()

```
constexpr int & Renderer::RayDepth ( ) [inline], [constexpr], [noexcept]
Definition at line 72 of file Renderer.h.
00073
              return m_RayDepth;
00074
3.12.4.11 Render() [1/2]
void Renderer::Render (
               const Camera & camera,
               const AccelerationStructure & accelerationStructure,
               std::span< const Light > lightSources,
               std::span< const Material > materials ) [noexcept]
Definition at line 112 of file Renderer.cpp.
00112
00113
          m_Camera = &camera;
          m_AccelerationStructure = &accelerationStructure;
00114
00115
          m_LightSources = lightSources;
00116
          m Materials = materials;
00117
00118
          if (!m_Accumulate)
00119
              m_FrameIndex = 1;
00120
          }
00121
00122
          if (m FrameIndex == 1) {
              memset(m_AccumulationData, 0, m_Width * m_Height * sizeof(Math::Vector4f));
00123
00125
00126
          float inverseFrameIndex = 1.f / m_FrameIndex;
00127
          float inverseGamma = 1.f / m_Gamma;
00128
00129
          std::vector<std::thread> handles;
00130
          handles.reserve(m_UsedThreads);
00131
00132
          for (int i = 0; i < m_Height; i += m_LinesPerThread) {</pre>
00133
              handles.emplace_back([this, i, inverseFrameIndex, inverseGamma]() {
                   int nextBlock = i + m_LinesPerThread;
00134
                   int limit = Math::Min(nextBlock, m_Height);
for (int t = i; t < limit; ++t) {</pre>
00135
00136
00137
                       for (int j = 0; j < m_Width; ++j) {</pre>
00138
                           \label{eq:m_AccumulationData} $$ m_AccumulationData[m_Width * t + j] += AcceleratedPixelProgram(t, j); $$
00139
00140
                           Math::Vector4f color = m_AccumulationData[m_Width * t + j];
00141
00142
                           color *= inverseFrameIndex;
00143
                           color = Utilities::CorrectGamma(color, inverseGamma);
00144
                           color = Math::Clamp(color, 0.f, 1.f);
00145
00146
                           \label{eq:m_mage} \verb|m_Midth * t + j| = \verb|Utilities::ConvertColorToRGBA(color);| \\
00147
00148
00149
              });
00150
          }
00151
00152
          for (auto &handle : handles) {
00153
              handle.join();
00154
00155
00156
          m_Image->UpdateData(m_ImageData);
00157
00158
          if (m Accumulate) {
00159
              ++m_FrameIndex;
00160
00161 }
```

#### 3.12.4.12 Render() [2/2]

void Renderer::Render (

```
const Camera & camera,
               std::span< const HittableObjectPtr > objects,
               std::span< const Light > lightSources,
               std::span< const Material > materials ) [noexcept]
Definition at line 61 of file Renderer.cpp.
00061
00062
          m_Camera = &camera;
00063
          m_Objects = objects;
00064
          m_LightSources = lightSources;
00065
          m Materials = materials;
00066
00067
          if (!m_Accumulate) +
00068
             m_FrameIndex = 1;
00069
00070
00071
          if (m_FrameIndex == 1) {
00072
             memset(m_AccumulationData, 0, m_Width * m_Height * sizeof(Math::Vector4f));
00073
00074
00075
          float inverseFrameIndex = 1.f / m_FrameIndex;
00076
          float inverseGamma = 1.f / m_Gamma;
00077
00078
          std::vector<std::thread> handles;
00079
          handles.reserve(m_UsedThreads);
08000
00081
          for (int i = 0; i < m_Height; i += m_LinesPerThread) {</pre>
00082
              handles.emplace_back([this, i, inverseFrameIndex, inverseGamma]() {
00083
                  int nextBlock = i + m_LinesPerThread;
                  int limit = Math::Min(nextBlock, m_Height);
for (int t = i; t < limit; ++t) {</pre>
00084
00085
00086
                      for (int j = 0; j < m_Width; ++j) {</pre>
00087
                           m_AccumulationData[m_Width * t + j] += PixelProgram(t, j);
00088
00089
                          Math::Vector4f color = m_AccumulationData[m_Width * t + j];
00090
00091
                          color *= inverseFrameIndex;
00092
                           color = Utilities::CorrectGamma(color, inverseGamma);
00093
                           color = Math::Clamp(color, 0.f, 1.f);
00094
                           m_ImageData[m_Width * t + j] = Utilities::ConvertColorToRGBA(color);
00095
00096
                      }
00097
                  }
00098
              });
00099
          }
00100
00101
          for (auto &handle : handles) {
00102
              handle.join();
00103
00104
00105
          m_Image->UpdateData(m_ImageData);
00106
00107
          if (m_Accumulate) {
00108
              ++m_FrameIndex;
00109
00110 }
3.12.4.13 SaveImage()
void Renderer::SaveImage (
               const char * filename ) const [noexcept]
```

 $\tt stbi\_write\_png(filename, m\_Width, m\_Height, 4, m\_ImageData, m\_Width * 4);$ 

#### Generated by Doxygen

00035

00037 }

Definition at line 35 of file Renderer.cpp.

32 Class Documentation

#### 3.12.4.14 SetUsedThreadCount()

```
constexpr void Renderer::SetUsedThreadCount (
              int usedThreads ) [inline], [constexpr], [noexcept]
Definition at line 59 of file Renderer.h.
00059
00060
             m_UsedThreads = Math::Clamp(usedThreads, 1, m_AvailableThreads);
00061
             m_LinesPerThread = (m_Height + m_UsedThreads - 1) / m_UsedThreads;
00062
3.12.4.15 UsedThreadCount()
constexpr int & Renderer::UsedThreadCount ( ) [inline], [constexpr], [noexcept]
Definition at line 64 of file Renderer.h.
                                                  {
00065
             return m_UsedThreads;
00066
```

The documentation for this class was generated from the following files:

- · Renderer.h
- · Renderer.cpp

#### 3.13 Scene Struct Reference

#### **Public Member Functions**

- std::optional< std::string > Serialize (std::ostream &os) const noexcept
- std::optional < std::string > Deserialize (std::istream &is) noexcept

#### **Public Attributes**

- std::vector< Shapes::Sphere > spheres
- std::vector< Shapes::Triangle > triangles
- std::vector < Shapes::Box > boxes
- std::vector< Model \* > models
- std::vector< Material > materials
- · Camera camera

#### 3.13.1 Detailed Description

Definition at line 17 of file Scene.h.

#### 3.13.2 Member Function Documentation

#### 3.13.2.1 Deserialize()

```
std::optional< std::string > Scene::Deserialize (
              std::istream & is ) [inline], [noexcept]
Definition at line 37 of file Scene.h.
00038
             is.exceptions(std::ios::eofbit | std::ios::badbit | std::ios::failbit);
00039
             try {
    TryDeserialize(is);

00040
00041
00042
             } catch (std::exception &e) {
00043
                 return e.what();
00044
00045
00046
             return {};
00047
```

#### 3.13.2.2 Serialize()

```
00025
             os.exceptions(std::ios::badbit | std::ios::failbit);
00027
00028
00029
                TrySerialize(os);
00030
             } catch (std::exception &e) {
00031
                return e.what();
00032
00033
00034
             return {};
       }
00035
```

#### 3.13.3 Member Data Documentation

#### 3.13.3.1 boxes

```
std::vector<Shapes::Box> Scene::boxes
```

Definition at line 20 of file Scene.h.

#### 3.13.3.2 camera

Camera Scene::camera

Definition at line 23 of file Scene.h.

#### 3.13.3.3 materials

```
std::vector<Material> Scene::materials
```

Definition at line 22 of file Scene.h.

34 Class Documentation

#### 3.13.3.4 models

```
std::vector<Model*> Scene::models
```

Definition at line 21 of file Scene.h.

#### 3.13.3.5 spheres

```
std::vector<Shapes::Sphere> Scene::spheres
```

Definition at line 18 of file Scene.h.

#### 3.13.3.6 triangles

```
std::vector<Shapes::Triangle> Scene::triangles
```

Definition at line 19 of file Scene.h.

The documentation for this struct was generated from the following file:

· Scene.h

#### 3.14 Timer Class Reference

#### **Public Member Functions**

- constexpr Timer (const Timer &)=delete
- constexpr Timer (Timer &&)=delete

#### **Static Public Member Functions**

• static double MeasureInMillis (std::function< void()> f)

#### 3.14.1 Detailed Description

Definition at line 7 of file Timer.h.

#### 3.14.2 Member Function Documentation

#### 3.14.2.1 MeasureInMillis()

The documentation for this class was generated from the following file:

· Timer.h

# **Chapter 4**

# **File Documentation**

#### 4.1 AABB.h

```
00001 #ifndef _AABB_H
00002 #define _AABB_H
00003
00004 #include "math/Math.h"
00005
00006 struct AABB {
00007
          Math:: Vector3f min, max;
00008
00009
          constexpr AABB() noexcept = default;
00010
          constexpr AABB(const Math::Vector3f &a, const Math::Vector3f &b) noexcept :
00011
00012
              min(Math::Min(a, b)), max(Math::Max(a, b)) {}
00013
00014
          constexpr AABB(const AABB &other) noexcept = default;
00015
00016
          constexpr AABB(const AABB &a, const AABB &b) noexcept:
00017
              min(Math::Min(a.min, b.min)), max(Math::Max(a.max, b.max)) {}
00018
00019
          constexpr float GetSurfaceArea() const noexcept {
            float a, b, c;
00021
              a = max.x - min.x;
              b = max.y - min.y;
c = max.z - min.z;
00022
00023
00024
              return 2.f * a * b + 2.f * b * c + 2.f * a * c;
00025
         }
00026
00027
          constexpr bool IntersectsRay(const Ray &ray, float t_min, float t_max) const noexcept {
00028
             auto one_over_direction = 1.f / ray.direction;
00029
              auto origin = ray.origin;
00030
              auto t0 = (min - origin) * one_over_direction;
auto t1 = (max - origin) * one_over_direction;
00031
00032
00033
00034
              if (one_over_direction.x < 0.f) {</pre>
00035
                   std::swap(t0.x, t1.x);
00036
00037
00038
              if (one_over_direction.y < 0.f) {</pre>
                   std::swap(t0.y, t1.y);
00040
00041
00042
              if (one_over_direction.z < 0.f) {</pre>
00043
                   std::swap(t0.z, t1.z);
00044
00045
00046
              t_min = Math::Max(t_min, Math::Max(Math::Max(t0.x, t0.y), t0.z));
00047
              t_{max} = Math::Min(t_{max}, Math::Min(Math::Min(t1.x, t1.y), t1.z));
00048
00049
              if (t_max <= t_min) {</pre>
00050
                   return false;
00051
00052
00053
              return true;
00054
         }
00055
00056
          constexpr static AABB Empty() noexcept {
00057
              AABB aabb;
              aabb.min = Math::Vector3f(+Math::Constants::Infinity<float>);
```

## 4.2 AccelerationStructure.cpp

```
00001 #include "AccelerationStructure.h" 00002 #include "Utilities.hpp"
00003 #include "hittable/NonHittable.h"
00005 AccelerationStructure::AccelerationStructure() noexcept {
00006
          m_Root = new BVHNode(new NonHittable());
00007 }
80000
00009 AccelerationStructure::AccelerationStructure(std::span<HittableObjectPtr> objects) noexcept {
00010
          Update (objects);
00011 }
00012
00013 AccelerationStructure::~AccelerationStructure() noexcept {
00014
          if (m_Root != nullptr) {
00015
              BVHNode::FreeMemory(m Root);
00016
00017 }
00018
00019 void AccelerationStructure::Update(std::span<HittableObjectPtr> objects) noexcept {
00020
        if (m_Root != nullptr) {
              BVHNode::FreeMemory(m_Root);
00021
00022
          }
00024
          if (objects.empty()) {
00025
              m_Root = new BVHNode(new NonHittable());
00026
          } else {
00027
              m_Root = BVHNode::MakeHierarchySAH(objects, 0, static_cast<int>(objects.size()));
00028
00029 }
00031 void AccelerationStructure::Hit(const Ray &ray, float tMin, float tMax, HitPayload &payload) const
     noexcept {
00032
          m_Root->Hit(ray, tMin, tMax, payload);
00033 }
```

#### 4.3 AccelerationStructure.h

```
00001 #ifndef _ACCELERATION_STRUCTURE_H
00002 #define _ACCELERATION_STRUCTURE_H
00003
00004 #include "hittable/HittableObject.h"
00005 #include "math/Math.h"
00006 #include "AABB.h"

00007 #include "Ray.h"

00008 #include "HitPayload.h"

00009 #include "BVHNode.h"
00010
00011 #include <span>
00012
00013 class AccelerationStructure {
00014 public:
00015
           AccelerationStructure() noexcept;
00016
00017
           AccelerationStructure(std::span<HittableObjectPtr> objects) noexcept;
00018
00019
           ~AccelerationStructure() noexcept;
00020
00021
           void Update(std::span<HittableObjectPtr> objects) noexcept;
00022
00023
           void Hit (const Ray &ray, float tMin, float tMax, HitPayload &payload) const noexcept;
00024
00025 private:
00026
           BVHNode *m_Root = nullptr;
00027 };
00028
00029 #endif
```

4.4 Application.cpp 37

## 4.4 Application.cpp

```
00001 #include "Application.h"
00002 #include "Utilities.hpp"
00003 #include "Timer.h"
00004
00005 #include "../imgui-docking/imgui.h"
00006 #include "../imgui-docking/backends/imgui_impl_glfw.h"
00007 #include "../imgui-docking/backends/imgui_impl_opengl3.h"
80000
00009 #define STB_IMAGE_IMPLEMENTATION
00010 #include "../stb-master/stb_image.h"
00012 #define GL_SILENCE_DEPRECATION
00013 #if defined(IMGUI_IMPL_OPENGL_ES2)
00014 #include <GLES2/gl2.h>
00015 #endif
00016
00017 #include <iostream>
00018 #include <chrono>
00019 #include <fstream>
00020
00021 Application::Application(int windowWidth, int windowHeight) noexcept :
00022
          m_InitialWindowWidth(windowWidth), m_InitialWindowHeight(windowHeight),
m_LastViewportWidth(-1), m_LastViewportHeight(-1),
00023
          m_Renderer(windowWidth, windowHeight),
m_TotalRenderTime(0.f), m_LastRenderTime(0.f) {
00024
00025
00026
00027
           m_SaveImageFilePath = new char[c_AnyInputFilePathLength];
          m_SceneFilePath = new char[c_AnyInputFilePathLength];
m_ModelFilePath = new char[c_AnyInputFilePathLength];
00028
00029
           m_MaterialDirectory = new char[c_AnyInputFilePathLength];
00031
           memset(m_SaveImageFilePath, 0, sizeof(m_SaveImageFilePath));
00032
           memset(m_SceneFilePath, 0, sizeof(m_SceneFilePath));
00033
           memset(m_ModelFilePath, 0, sizeof(m_ModelFilePath));
00034
           {\tt memset} \; ({\tt m\_Material Directory}, \;\; {\tt 0, } \;\; {\tt sizeof} \; ({\tt m\_Material Directory}) \, ) \; ; \\
00035
00036
           m_AddMaterial.albedo = {0.f, 0.f, 0.f};
00037
           m_AddMaterial.metallic = 0.f;
00038
           m_AddMaterial.specular = 0.f;
00039
           m_AddMaterial.roughness = 0.f;
00040
           m_AddMaterial.emissionPower = 0.f;
00041
           m AddMaterial.index = -1:
00042
00043
           m_AddSphere = Shapes::Sphere(Math::Vector3f(0.f), 0.f, nullptr);
00044
           m_AddTriangle = Shapes::Triangle(Math::Vector3f(0.f), Math::Vector3f(0.f), Math::Vector3f(0.f),
00045
           m_AddBox = Shapes::Box(Math::Vector3f(0.f), Math::Vector3f(0.f), nullptr);
00046
00047
           m AddSphereMaterialIndex = -1;
00048
           m_AddTriangleMaterialIndex = -1;
00049
           m_AddBoxMaterialIndex = -1;
00050
00051
           m_RayMissColor = Math::Vector3f(0.f);
00052
00053
           m Scene.camera = Camera(windowWidth, windowHeight);
00054
00055
           LoadSceneFromFile(c_DefaultScenePath);
00056 }
00057
00058 int Application::Run() noexcept {
00059
           if (!alfwInit())
               std::cerr « "Failed to initialize GLFW\n";
00060
00061
               return -1;
00062
00063
00064 #if defined(IMGUI_IMPL_OPENGL_ES2)
          const char* glsl_version = "#version 100";
00065
00066
           glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 2);
00067
           glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 0);
00068
           glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_ES_API);
00069 #elif defined(__APPLE___
           const char* glsl_version = "#version 150";
00070
00071
           glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
           glfwWindowHint (GLFW_CONTEXT_VERSION_MINOR, 2);
00072
           glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
00074
           glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
00075 #else
00076
           const char* glsl_version = "#version 130";
           glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
00077
           glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 0);
00078
00079 #endif
00080
00081
           m_Window = glfwCreateWindow(m_InitialWindowWidth, m_InitialWindowHeight, c_WindowTitle, nullptr,
      nullptr);
00082
           if (m Window == nullptr) {
               std::cerr « "Failed to create window\n";
00083
```

```
00084
              return -1;
00085
00086
00087
          GLFWimage *icon = new GLFWimage();
          icon->pixels = stbi_load("icon/icon.png", &icon->width, &icon->height, 0, 4);
00088
          glfwSetWindowIcon(m_Window, 1, icon);
00089
00090
          stbi_image_free(icon->pixels);
00091
          delete icon;
00092
00093
          glfwMakeContextCurrent(m_Window);
00094
00095
          ImGui::CreateContext();
00096
          ImGuiIO &io = ImGui::GetIO();
00097
          io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;
00098
00099
          UpdateThemeStyle();
00100
00101
           ImGui ImplGlfw InitForOpenGL(m Window, true);
          ImGui_ImplOpenGL3_Init(glsl_version);
00102
00103
00104
00105
00106
          ImGui_ImplOpenGL3_Shutdown();
00107
          ImGui ImplGlfw Shutdown();
00108
          ImGui::DestroyContext();
00109
00110
          glfwDestroyWindow(m_Window);
00111
          glfwTerminate();
00112
00113
          return 0:
00114 }
00115
00116 void Application::MainLoop() noexcept {
00117
         while (!glfwWindowShouldClose(m_Window)) {
00118
              glfwPollEvents();
00119
00120
              ImGui ImplOpenGL3 NewFrame();
              ImGui_ImplGlfw_NewFrame();
00121
00122
              ImGui::NewFrame();
00123
00124
              const ImGuiViewport* viewport = ImGui::GetMainViewport();
00125
              ImGui::SetNextWindowPos(viewport->WorkPos);
00126
00127
               ImGui::SetNextWindowSize(viewport->WorkSize);
00128
              ImGui::SetNextWindowViewport(viewport->ID);
00129
00130
              ImGuiWindowFlags dockWindowFlags = ImGuiWindowFlags_NoDocking;
     dockWindowFlags |= ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoCollapse | ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoMove;
00131
00132
              dockWindowFlags |= ImGuiWindowFlags_NoBringToFrontOnFocus | ImGuiWindowFlags_NoNavFocus;
00133
00134
00135
                   OnUpdate();
00136
              }
00137
00138
              ImGui::Render();
00139
              ImGui::UpdatePlatformWindows();
00140
00141
              int display_w, display_h;
00142
               glfwGetFramebufferSize(m_Window, &display_w, &display_h);
              glViewport(0, 0, display_w, display_h);
glClearColor(0.0, 0.0, 0.0, 1.0);
00143
00144
00145
              glClear(GL_COLOR_BUFFER_BIT);
00146
00147
              ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
00148
00149
              glfwSwapBuffers(m_Window);
00150
          }
00151 }
00152
00153 void Application::OnUpdate() noexcept {
00154
          const ImGuiViewport* viewport = ImGui::GetMainViewport();
          ImGuiWindowFlags frameFlags = ImGuiViewportFlags_IsPlatformWindow
00155
                                        | ImGuiViewportFlags_NoDecoration
00156
                                          ImGuiViewportFlags_NoTaskBarIcon
00157
00158
                                          ImGuiViewportFlags_NoAutoMerge
00159
                                          ImGuiWindowFlags_NoCollapse
00160
                                        | ImGuiWindowFlags_NoResize;
00161
00162
          ImGui::SetNextWindowPos(viewport->WorkPos);
          ImGui::SetNextWindowSize({viewport->WorkSize.x * 0.7f, viewport->WorkSize.y});
00163
00164
          ImGui::SetNextWindowViewport(viewport->ID);
00165
00166
          ImGui::Begin("Frame", nullptr, frameFlags);
00167
               int viewportWidth = (int) ImGui::GetContentRegionAvail().x;
00168
00169
               int viewportHeight = (int) ImGui::GetContentRegionAvail().y;
```

4.4 Application.cpp 39

```
int width = -1, height = -1;
00171
00172
               glfwGetWindowSize(m_Window, &width, &height);
00173
00174
               if ((m_LastViewportWidth != viewportWidth || m_LastViewportHeight != viewportHeight) && width
      * height > 0) {
00175
                   m_LastViewportWidth = viewportWidth;
00176
                   m_LastViewportHeight = viewportHeight;
00177
00178
                   \verb|m_Scene.camera.OnV| is wportResize (m_LastV| is wportW| idth, m_LastV| is wportHeight); \\
00179
00180
                   m Renderer.OnResize(m LastViewportWidth, m LastViewportHeight);
00181
               }
00182
00183
               Image *image = m_Renderer.GetImage();
00184
               if (image != nullptr)
00185
                   ImGui::Image((void*)(intptr_t)image->GetDescriptor(), ImGui::GetContentRegionAvail());
00186
               }
00187
00188
          ImGui::End();
00189
00190
           ImGui::SetNextWindowPos({viewport->WorkPos.x + viewport->WorkSize.x * 0.7f, viewport->WorkPos.y});
00191
           ImGui::SetNextWindowSize({viewport->WorkSize.x * 0.3f, viewport->WorkSize.y});
00192
          ImGui::SetNextWindowViewport(viewport->ID);
00193
00194
           ImGui::Begin("Options", nullptr, ImGuiWindowFlags_NoTitleBar);
00195
               m_{\text{LastID}} = 0;
00196
00197
               m_SomeObjectChanged = false;
00198
               m_SomeGeometryChanged = false;
00199
00200
               ProcessSceneCollapsingHeaders();
00201
               ImGui::Checkbox("Accumulate", Math::ValuePointer(m_Renderer.Accumulate()));
ImGui::Checkbox("Accelerate", Math::ValuePointer(m_Renderer.Accelerate()));
if (ImGui::InputInt("Used threads", Math::ValuePointer(m_Renderer.UsedThreadCount()))) {
00202
00203
00204
00205
                   m_Renderer.SetUsedThreadCount (m_Renderer.UsedThreadCount());
00206
00207
               ImGui::InputInt("Ray depth", Math::ValuePointer(m_Renderer.RayDepth()));
00208
               ImGui::InputFloat("Gamma", Math::ValuePointer(m_Renderer.Gamma()));
00209
00210
               if (ImGui::ColorEdit3("Ray miss color", Math::ValuePointer(m_RayMissColor))) {
                   m_Renderer.OnRayMiss([this](const Ray&){ return m_RayMissColor; });
00211
00212
00213
00214
               if (ImGui::Button("Reset", {viewport->WorkSize.x * 0.05f, viewport->WorkSize.y * 0.1f}) ||
      m_Renderer.Accumulate()) {
00215
                   m_Scene.camera.ComputeRayDirections();
00216
00217
                   m LastRenderTime = Timer::MeasureInMillis([this]() {
00218
                       if (m_Renderer.Accelerate()) {
00219
                            m_Renderer.Render(m_Scene.camera, m_AccelerationStructure, m_Lights,
      m_Scene.materials);
                      } else {
00220
00221
                            m_Renderer.Render(m_Scene.camera, m_Objects, m_Lights, m_Scene.materials);
00222
00223
                   });
00224
00225
                   if (m_Renderer.Accumulate()) {
00226
                       m_TotalRenderTime += m_LastRenderTime;
00227
                   } else {
00228
                       m TotalRenderTime = 0.f;
00229
                   }
00230
              }
00231
00232
               if (ImGui::Checkbox("Dark theme", Math::ValuePointer(m_DarkTheme))) {
00233
                   UpdateThemeStyle();
00234
              }
00235
00236
               ImGui::InputText("##save_image", m_SaveImageFilePath, c_AnyInputFilePathLength);
00237
               ImGui::SameLine();
               if (ImGui::Button("Save image")) {
00238
00239
                   m_Renderer.SaveImage(m_SaveImageFilePath);
00240
00241
00242
               ImGui::InputText("##save_scene", m_SceneFilePath, c_AnyInputFilePathLength);
               if (ImGui::Button("Save scene")) {
00243
00244
                   SaveSceneToFile(m_SceneFilePath);
00245
00246
00247
               ImGui::SameLine();
               if (ImGui::Button("Load scene")) {
00248
                   LoadSceneFromFile(m_SceneFilePath);
00249
00250
00251
               ImGui::Text("Last render time: %fms", m_LastRenderTime);
00252
00253
               ImGui::Text("Average render time: %fms", m TotalRenderTime /
```

```
(Math::Max(m_Renderer.GetFrameIndex() - 1, 1)));
00254
              ImGui::Text("Accumulated frame count: %d", Math::Max(m_Renderer.GetFrameIndex() - 1, 1));
00255
00256
00257
          TmGui::End():
00258 }
00259
00260 void Application::UpdateThemeStyle() noexcept {
00261
          if (m_DarkTheme) {
00262
               ImGui::StyleColorsDark();
00263
          } else {
00264
               ImGui::StvleColorsLight();
00265
          }
00266 }
00267
00268 void Application::ProcessSceneCollapsingHeaders() noexcept {
00269
          ProcessCameraCollapsingHeader();
          ProcessSpheresCollapsingHeader();
00270
          ProcessTrianglesCollapsingHeader();
00271
00272
          ProcessBoxesCollapsingHeader();
00273
          ProcessModelsCollapsingHeader();
00274
          ProcessMaterialsCollapsingHeader();
00275
00276
          if (m SomeObjectChanged) {
00277
               UpdateObjects();
00278
          }
00279
00280
          if (m_SomeGeometryChanged) {
00281
               m_AccelerationStructure.Update(m_Objects);
00282
00283 }
00284
00285 void Application::ProcessCameraCollapsingHeader() noexcept {
00286
          ImGui::PushID(m_LastID++);
00287
          if (ImGui::CollapsingHeader("Camera", nullptr)) {
00288
               bool cameraNeedUpdate = false;
ImGui::InputFloat3("Position", Math::ValuePointer(m_Scene.camera.Position()));
00289
00290
               ImGui::InputFloat3("Target", Math::ValuePointer(m_Scene.camera.Target()));
ImGui::InputFloat("Vertical FOV", Math::ValuePointer(m_Scene.camera.VerticalFovInDegrees()));
00291
00292
00293
               ImGui::InputFloat3("Up", Math::ValuePointer(m_Scene.camera.Up()));
00294
          }
00295
00296
          ImGui::PopID();
00297 }
00298
00299 void Application::ProcessSpheresCollapsingHeader() noexcept {
00300
          int deleteIndex = -1;
          if (ImGui::CollapsingHeader("Spheres", nullptr)) {
00301
00302
               for (int i = 0; i < (int)m_Scene.spheres.size(); ++i) {</pre>
00303
                   ImGui::PushID(m_LastID++);
00304
00305
                   Shapes::Sphere &sphere = m_Scene.spheres[i];
                   ImGui::Text("Sphere %d:", i);
00306
00307
00308
                   if (ImGui::Button("Delete")) {
00309
                       deleteIndex = i;
00310
00311
                   if (ImGui::InputFloat("Radius", Math::ValuePointer(sphere.radius))) {
00312
00313
                       sphere = Shapes::Sphere(sphere.center, sphere.radius, sphere.material);
00314
                       m_SomeGeometryChanged = true;
00315
                   }
00316
00317
                   if (ImGui::InputFloat3("Position", Math::ValuePointer(sphere.center))) {
00318
                       m_SomeGeometryChanged = true;
00319
                   }
00320
00321
                   if (ImGui::InputInt("Material index", Math::ValuePointer(m_SphereMaterialIndices[i]))) {
00322
                       sphere.material = &m_Scene.materials[m_SphereMaterialIndices[i]];
00323
00324
00325
                   ImGui::PopID();
00326
               }
00327
00328
               ImGui::PushID(m LastID++);
00329
00330
               if (ImGui::Button("Add")) {
00331
                   \verb|m_Scene.spheres.push_back(m_AddSphere)|;
00332
                   m_SomeObjectChanged = true;
00333
                   m_SomeGeometryChanged = true;
00334
               }
00335
00336
               if (ImGui::InputFloat("Radius", Math::ValuePointer(m_AddSphere.radius))) {
00337
                   m_AddSphere = Shapes::Sphere(m_AddSphere.center, m_AddSphere.radius,
      m_AddSphere.material);
00338
```

4.4 Application.cpp 41

```
00339
00340
                                   ImGui::InputFloat3("Position", Math::ValuePointer(m AddSphere.center));
00341
                                   if (ImGui::InputInt("Material index", Math::ValuePointer(m_AddSphereMaterialIndex))) {
00342
00343
                                             m_AddSphere.material = &m_Scene.materials[m_AddSphereMaterialIndex];
00344
00345
00346
                                   ImGui::PopID();
00347
00348
                                   if (deleteIndex >= 0) {
00349
                                            m_Scene.spheres.erase(m_Scene.spheres.cbegin() + deleteIndex);
00350
                                             m SomeObjectChanged = true;
00351
                                             m_SomeGeometryChanged = true;
00352
00353
                         }
00354 }
00355
00356 void Application::ProcessTrianglesCollapsingHeader() noexcept {
                       int deleteIndex = -1;
00358
                         if (ImGui::CollapsingHeader("Triangles", nullptr)) {
00359
                                   for (int i = 0; i < (int)m_Scene.triangles.size(); ++i) {</pre>
00360
                                            ImGui::PushID(m_LastID++);
00361
00362
                                            Shapes::Triangle &triangle = m_Scene.triangles[i];
ImGui::Text("Triangle %d:", i);
00363
00364
00365
                                             if (ImGui::Button("Delete")) {
00366
                                                       deleteIndex = i;
00367
                                             }
00368
                                             if (ImGui::InputFloat3("Vertex 0", Math::ValuePointer(triangle.vertices[0]))) {
00369
00370
                                                       triangle = Shapes::Triangle(triangle.vertices[0], triangle.vertices[1],
              triangle.vertices[2], triangle.material);
00371
                                                       m_SomeGeometryChanged = true;
00372
                                             if (ImGui::InputFloat3("Vertex 1", Math::ValuePointer(triangle.vertices[1]))) {
00373
                                                       triangle = Shapes::Triangle(triangle.vertices[0], triangle.vertices[1],
00374
              triangle.vertices[2], triangle.material);
00375
                                                      m_SomeGeometryChanged = true;
00376
00377
                                             if (ImGui::InputFloat3("Vertex 2", Math::ValuePointer(triangle.vertices[2]))) {
                                                       triangle = Shapes::Triangle(triangle.vertices[0], triangle.vertices[1],
00378
              triangle.vertices[2], triangle.material);
00379
                                                      m_SomeGeometryChanged = true;
00380
                                             }
00381
00382
                                             if (ImGui::InputInt("Material index", Math::ValuePointer(m_TriangleMaterialIndices[i]))) {
00383
                                                        triangle.material = &m_Scene.materials[m_TriangleMaterialIndices[i]];
                                             }
00384
00385
00386
                                            ImGui::PopID();
00387
00388
00389
                                   ImGui::PushID(m_LastID++);
00390
00391
                                   if (ImGui::Button("Add")) {
00392
                                            m_Scene.triangles.push_back(m_AddTriangle);
00393
                                             m_SomeObjectChanged = true;
00394
                                             m_SomeGeometryChanged = true;
00395
                                  }
00396
                                   if (ImGui::InputFloat3("Vertex 0", Math::ValuePointer(m_AddTriangle.vertices[0]))) {
00397
00398
                                            m_AddTriangle = Shapes::Triangle(m_AddTriangle.vertices[0], m_AddTriangle.vertices[1],
              m_AddTriangle.vertices[2], m_AddTriangle.material);
00399
00400
                                   if (ImGui::InputFloat3("Vertex 1", Math::ValuePointer(m_AddTriangle.vertices[1]))) {
00401
                                            \verb|m_AddTriangle = Shapes:: Triangle (\verb|m_AddTriangle.vertices[0], \verb|m_AddTriangle.vertices[1], \verb|m_Ad
              m_AddTriangle.vertices[2], m_AddTriangle.material);
00402
00403
                                   if (ImGui::InputFloat3("Vertex 2", Math::ValuePointer(m_AddTriangle.vertices[2]))) {
                                              \texttt{m\_AddTriangle} = \texttt{Shapes::Triangle} \\  (\texttt{m\_AddTriangle.vertices[0]}, \\  \texttt{m\_AddTriangle.vertices[1]}, \\  \texttt{m\_AddTriangle.vert
00404
              m_AddTriangle.vertices[2], m_AddTriangle.material);
00405
                                  }
00406
00407
                                   if (ImGui::InputInt("Material index", Math::ValuePointer(m_AddTriangleMaterialIndex))) {
00408
                                             m_AddTriangle.material = &m_Scene.materials[m_AddTriangleMaterialIndex];
00409
                                   }
00410
00411
                                   ImGui::PopID();
00412
                                   if (deleteIndex >= 0) {
00413
00414
                                            m_Scene.triangles.erase(m_Scene.triangles.cbegin() + deleteIndex);
                                             m_SomeObjectChanged = true;
00415
00416
                                             m_SomeGeometryChanged = true;
00417
                                   }
00418
                        }
00419 }
```

```
00421 void Application::ProcessBoxesCollapsingHeader() noexcept {
          int deleteIndex = -1;
00422
          if (ImGui::CollapsingHeader("Boxes", nullptr)) {
00423
00424
               for (int i = 0; i < (int)m_Scene.boxes.size(); ++i) {</pre>
00425
                    ImGui::PushID(m_LastID++);
00427
                    ImGui::Text("Box: %d", i);
00428
00429
                    if (ImGui::Button("Delete")) {
00430
                        deleteIndex = i;
00431
00432
00433
                    Shapes::Box &box = m_Scene.boxes[i];
00434
00435
                    if (ImGui::InputFloat3("First corner", Math::ValuePointer(box.min))) {
00436
                        box = Shapes::Box(box.min, box.max, box.material);
00437
                        m_SomeGeometryChanged = true;
00438
00439
                   if (ImGui::InputFloat3("Second corner", Math::ValuePointer(box.max))) {
   box = Shapes::Box(box.min, box.max, box.material);
00440
00441
00442
                        m_SomeGeometryChanged = true;
00443
                    }
00444
00445
                    if (ImGui::InputInt("Material index", Math::ValuePointer(m_BoxMaterialIndices[i]))) {
00446
                        box = Shapes::Box(box.min, box.max, &m_Scene.materials[m_BoxMaterialIndices[i]]);
00447
00448
00449
                   ImGui::PopID();
00450
00451
00452
               ImGui::PushID(m_LastID++);
00453
00454
               if (ImGui::Button("Add")) {
00455
                   m_Scene.boxes.push_back(m_AddBox);
00456
                    m SomeObjectChanged = true;
                   m_SomeGeometryChanged = true;
00458
               }
00459
00460
               if (ImGui::InputFloat3("First corner", Math::ValuePointer(m_AddBox.min))) {
00461
                    m_AddBox = Shapes::Box(m_AddBox.min, m_AddBox.max, m_AddBox.material);
00462
00463
               if (ImGui::InputFloat3("Second corner", Math::ValuePointer(m_AddBox.max))) {
00464
                   m_AddBox = Shapes::Box(m_AddBox.min, m_AddBox.max, m_AddBox.material);
00465
               }
00466
               if (ImGui::InputInt("Material index", Math::ValuePointer(m_AddBoxMaterialIndex))) {
00467
00468
                    m_AddBox.material = &m_Scene.materials[m_AddBoxMaterialIndex];
00469
               }
00470
00471
               ImGui::PopID();
00472
00473
               if (deleteIndex >= 0) {
00474
                   m_Scene.boxes.erase(m_Scene.boxes.cbegin() + deleteIndex);
00475
                   m SomeObjectChanged = true;
                   m_SomeGeometryChanged = true;
00476
00477
               }
00478
          }
00479 }
00480
00481 void Application::ProcessModelsCollapsingHeader() noexcept {
00482
          int deleteIndex = -1;
           if (ImGui::CollapsingHeader("Models", nullptr))
00483
00484
                for (int i = 0; i < (int)m_Scene.models.size(); ++i) {</pre>
00485
                   ImGui::PushID(m_LastID++);
00486
00487
                   auto model = m Scene.models[i];
00488
00489
                   ImGui::Text("Model: %d", i);
00490
00491
                    auto materials = model->GetMaterials();
                    for (int i = 0; i < (int)materials.size(); ++i) {</pre>
00492
                        ImGui::PushID(m LastID++);
00493
00494
00495
                        Material &material = materials[i];
00496
                        ImGui::Text("Material %d:", material.index);
00497
                        ImGui::ColorEdit3("Albedo", Math::ValuePointer(material.albedo));
ImGui::InputFloat("Emission power", Math::ValuePointer(material.emissionPower));
ImGui::InputFloat("Metallic", Math::ValuePointer(material.metallic));
ImGui::InputFloat("Roughness", Math::ValuePointer(material.roughness));
00498
00499
00500
00501
00502
                        ImGui::InputFloat("Specular", Math::ValuePointer(material.specular));
00503
00504
                        ImGui::PopID();
00505
                    }
00506
```

4.4 Application.cpp 43

```
if (ImGui::Button("Delete")) {
00508
                         deleteIndex = i;
00509
                     }
00510
00511
                     ImGui::PopID();
00512
                }
00513
00514
                ImGui::PushID(m_LastID++);
00515
00516
                if (ImGui::Button("Import")) {
                     auto result = Model::LoadOBJ(m_ModelFilePath, m_MaterialDirectory);
00517
00518
00519
                     if (!result.warning.empty()) {
                          std::cout « "Warnings occured while loading model " « m_ModelFilePath « " with
      material directory: " « m_MaterialDirectory « ": " « result.warning « '\n';
00521
00522
00523
                    if (result.IsFailure()) {
                         std::cerr « "Failed to import model " « m_ModelFilePath « " with material directory: "
00524
      « m_MaterialDirectory « '\n';
00525
                   } else {
00526
                         std::cout « "Loaded model " « m_ModelFilePath « " with material directory: " «
      m_MaterialDirectory « '\n';
00527
00528
                         m_Scene.models.push_back(result.model);
00529
                         m_SomeObjectChanged = true;
00530
                         m_SomeGeometryChanged = true;
00531
                    }
00532
                }
00533
00534
                ImGui::InputText("Path (.obj)", m_ModelFilePath, c_AnyInputFilePathLength);
00535
                ImGui::InputText("Material folder", m_MaterialDirectory, c_AnyInputFilePathLength);
00536
00537
                ImGui::PopID();
00538
                if (deleteIndex >= 0) {
00539
00540
                    delete m Scene.models[deleteIndex];
00541
                    m_Scene.models.erase(m_Scene.models.cbegin() + deleteIndex);
00542
00543
                    m_SomeObjectChanged = true;
00544
                     m_SomeGeometryChanged = true;
00545
                }
00546
           }
00547 }
00548
00549 void Application::ProcessMaterialsCollapsingHeader() noexcept {
00550
         int deleteIndex = -1;
           if (ImGui::CollapsingHeader("Materials", nullptr)) {
00551
00552
                for (int i = 0; i < (int)m_Scene.materials.size(); ++i) {
   ImGui::PushID(m_LastID++);</pre>
00553
00554
00555
                     Material &material = m_Scene.materials[i];
00556
                     ImGui::Text("Material %d:", material.index);
00557
00558
                     if (ImGui::Button("Delete")) {
00559
                          deleteIndex = i;
00561
00562
                     ImGui::ColorEdit3("Albedo", Math::ValuePointer(material.albedo));
                    ImGui::InputFloat("Emission power", Math::ValuePointer(material.emissionPower));
ImGui::InputFloat("Metallic", Math::ValuePointer(material.emissionPower));
ImGui::InputFloat("Roughness", Math::ValuePointer(material.roughness));
ImGui::InputFloat("Specular", Math::ValuePointer(material.specular));
00563
00564
00565
00566
00567
00568
                    ImGui::PopID();
00569
                }
00570
00571
                ImGui::PushID(m LastID++);
00572
                if (ImGui::Button("Add")) {
00574
                    int maxIndex = -1;
00575
                     for (const auto &material : m_Scene.materials) {
00576
                         maxIndex = Math::Max(maxIndex, material.index);
00577
00578
00579
                     m_AddMaterial.index = maxIndex + 1;
00580
                     m_Scene.materials.push_back(m_AddMaterial);
00581
                     UpdateObjectMaterials();
00582
00583
00584
                ImGui::ColorEdit3("Albedo", Math::ValuePointer(m_AddMaterial.albedo));
                ImGui::InputFloat("Emission power", Math::ValuePointer(m_AddMaterial.emissionPower));
ImGui::InputFloat("Metallic", Math::ValuePointer(m_AddMaterial.metallic));
ImGui::InputFloat("Roughness", Math::ValuePointer(m_AddMaterial.roughness));
00585
00586
00587
                ImGui::InputFloat("Specular", Math::ValuePointer(m_AddMaterial.specular));
00588
00589
00590
                ImGui::PopID();
```

```
00592
              if (deleteIndex >= 0) {
00593
                  m_Scene.materials.erase(m_Scene.materials.cbegin() + deleteIndex);
00594
00595
00596 }
00597
00598 void Application::LoadSceneFromFile(const std::filesystem::path &pathToFile) noexcept {
00599
          std::ifstream fileStream(pathToFile, std::ios::binary);
          if (!fileStream) {
    std::cerr « "Failed to open file: " « pathToFile « '\n';
00600
00601
00602
              return:
00603
          }
00604
00605
          auto error = m_Scene.Deserialize(fileStream);
00606
          if (error.has_value()) {
              std::cerr « "Failed to deserialize scene: " « pathToFile « '\n';
00607
00608
              return;
          } else {
00609
00610
             std::cout « "Loaded scene: " « pathToFile « '\n';
00611
00612
00613
          UpdateObjects();
00614
          UpdateLights();
00615
          m_AccelerationStructure.Update(m_Objects);
00616 }
00617
00618 void Application::SaveSceneToFile(const std::filesystem::path &pathToFile) const noexcept {
00619
          std::ofstream fileStream(pathToFile, std::ios::binary);
00620
          if (!fileStream) {
              std::cerr « "Failed to open file: " « pathToFile « '\n';
00621
00622
              return;
00623
00624
00625
          auto error = m_Scene.Serialize(fileStream);
00626
          if (error.has_value()) {
00627
              std::cerr « "Failed to serialize scene: " « pathToFile « '\n';
00628
00629 }
00630
00631 void Application::UpdateObjects() noexcept {
00632
          m_Objects.clear();
00633
00634
          m_SphereMaterialIndices.clear();
          for (auto &sphere : m_Scene.spheres) {
00635
00636
              m_Objects.push_back(&sphere);
00637
              m_SphereMaterialIndices.push_back(sphere.material->index);
00638
00639
00640
          m TriangleMaterialIndices.clear();
00641
          for (auto &triangle : m_Scene.triangles) {
00642
              m_Objects.push_back(&triangle);
00643
              m_TriangleMaterialIndices.push_back(triangle.material->index);
00644
          }
00645
00646
          m BoxMaterialIndices.clear();
00647
          for (auto &box : m_Scene.boxes) {
              m_Objects.push_back(&box);
00648
00649
              m_BoxMaterialIndices.push_back(box.material->index);
00650
          }
00651
00652
          for (auto model : m Scene.models) {
00653
              m_Objects.push_back(model);
00654
00655 }
00656
00657 void Application::UpdateLights() noexcept {
00658
          m_Lights.clear();
00659
00660
          for (auto &sphere : m_Scene.spheres) {
00661
              if (sphere.material->emissionPower > 0.f) {
00662
                  m_Lights.emplace_back(&sphere, sphere.material->GetEmission());
00663
              }
00664
          }
00665
00666
          for (auto &triangle : m_Scene.triangles) {
00667
              if (triangle.material->emissionPower > 0.f) {
00668
                  m_Lights.emplace_back(&triangle, triangle.material->GetEmission());
00669
              }
00670
          }
00671
00672
          for (auto &box : m_Scene.boxes) {
00673
              if (box.material->emissionPower > 0.f) {
00674
                  m_Lights.emplace_back(&box, box.material->GetEmission());
00675
00676
          }
00677
```

4.5 Application.h

```
for (auto model : m_Scene.models) {
00679
                 auto lights = model->GetLightSources();
00680
                 m_Lights.insert(m_Lights.cend(), lights.begin(), lights.end());
00681
00682 }
00683
00684 void Application::UpdateObjectMaterials() noexcept {
00685
            int sphereCount = static_cast<int>(m_Scene.spheres.size());
            for (int i = 0; i < sphereCount; ++i) {
   m_Scene.spheres[i].material = &*std::find_if(m_Scene.materials.cbegin(),</pre>
00686
00687
      m_Scene.materials.cend(), [this, i](const auto &material) {
00688
                      return material.index == m_SphereMaterialIndices[i];
00689
00690
00691
            int triangleCount = static_cast<int>(m_Scene.triangles.size());
for (int i = 0; i < triangleCount; ++i) {
    m_Scene.triangles[i].material = &*std::find_if(m_Scene.materials.cbegin(),</pre>
00692
00693
00694
      m_Scene.materials.cend(), [this, i](const auto &material) {
00695
                      return material.index == m_TriangleMaterialIndices[i];
00696
00697
            }
00698
           int boxCount = static_cast<int>(m_Scene.boxes.size());
for (int i = 0; i < boxCount; ++i) {
    m_Scene.boxes[i].material = &*std::find_if(m_Scene.materials.cbegin(),</pre>
00699
00700
      m_Scene.materials.cend(), [this, i](const auto &material)
00702
                     return material.index == m_BoxMaterialIndices[i];
00703
                 });
00704
            }
00705 }
```

## 4.5 Application.h

```
00001 #ifndef _APPLICATION_H
00002 #define _APPLICATION_H
00003
00004 #include "../glfw-3.4/include/GLFW/glfw3.h"
00005
00006 #include "Scene.h"
00007 #include "Camera.h"

00008 #include "AccelerationStructure.h"

00009 #include "Renderer.h"
00010
00011 #include <cstring>
00012 #include <filesystem>
00013
00015 class Application {
00016 public:
          Application(int windowWidth, int windowHeight) noexcept;
00017
00018
00019
          int Run() noexcept;
00020
00021 private:
00022
          void MainLoop() noexcept;
00023
00024
          void OnUpdate() noexcept;
00025
00026
          void ProcessSceneCollapsingHeaders() noexcept;
00027
00028
          void ProcessCameraCollapsingHeader() noexcept;
00029
00030
          void ProcessSpheresCollapsingHeader() noexcept;
00031
00032
          void ProcessTrianglesCollapsingHeader() noexcept;
00033
00034
          void ProcessBoxesCollapsingHeader() noexcept;
00035
00036
          void ProcessModelsCollapsingHeader() noexcept;
00037
00038
          void ProcessMaterialsCollapsingHeader() noexcept;
00039
00040
          void UpdateThemeStyle() noexcept;
00041
00042
          void LoadSceneFromFile(const std::filesystem::path &pathToFile) noexcept;
00043
00044
          void SaveSceneToFile(const std::filesystem::path &pathToFile) const noexcept;
00045
00046
          void UpdateObjects() noexcept;
00047
00048
          void UpdateLights() noexcept;
00049
00050
          void UpdateObjectMaterials() noexcept;
```

```
00051
00052 private:
00053
           int m_InitialWindowWidth, m_InitialWindowHeight;
00054
           int m\_LastViewportWidth, m\_LastViewportHeight;
00055
           GLFWwindow *m Window = nullptr;
00056
           bool m_DarkTheme = false;
00058
           int m_LastID;
00059
           bool m_SomeObjectChanged;
00060
           bool m_SomeGeometryChanged;
00061
00062
           double m TotalRenderTime;
00063
           double m LastRenderTime;
00064
00065
           char *m_SaveImageFilePath;
00066
           char *m_SceneFilePath;
00067
00068
           Scene m Scene;
00069
           AccelerationStructure m_AccelerationStructure;
00070
           Renderer m_Renderer;
00071
00072
           Math::Vector3f m_RayMissColor;
00073
00074
           std::vector<int> m_SphereMaterialIndices;
std::vector<int> m_TriangleMaterialIndices;
00075
00076
           std::vector<int> m_BoxMaterialIndices;
00077
00078
           std::vector<HittableObject*> m_Objects;
00079
           std::vector<Light> m_Lights;
08000
00081
           Material m AddMaterial:
00082
           Shapes::Sphere m_AddSphere;
00083
           Shapes::Triangle m_AddTriangle;
00084
           Shapes::Box m_AddBox;
00085
           char *m_ModelFilePath;
00086
           char *m_MaterialDirectory;
00087
           int m_AddSphereMaterialIndex;
00089
           int m_AddTriangleMaterialIndex;
00090
           int m_AddBoxMaterialIndex;
00091
           constexpr static const char *c_WindowTitle = "Path Tracing";
constexpr static const char *c_DefaultScenePath = "assets/dft2.scn";
00092
00093
           constexpr static int c_AnyInputFilePathLength = 128;
00094
00095 };
00096
00097 #endif
```

## 4.6 BSDF.cpp

```
00001 #include "BSDF.h"
00002 #include "Utilities.hpp"
00003
00004 Math::Vector3f BSDF::Sample(const Ray &ray, const HitPayload &payload, Math::Vector3f &throughput)
      noexcept {
          float diffuseRatio = 0.5f * (1.f - m_Material->metallic); float specularRatio = 1.f - diffuseRatio;
00005
00006
00007
80000
          Math::Vector3f V = -ray.direction;
00009
00010
          Math:: Vector3f reflectionDirection;
00011
          if (Utilities::RandomFloatInZeroToOne() < diffuseRatio) {</pre>
00012
               reflectionDirection = Utilities::RandomInHemisphere(payload.normal);
           } else {
00014
              Math::Vector3f halfVec;
00015
               {
00016
                   Math::Vector2f Xi{Utilities::RandomFloatInZeroToOne(),
      Utilities::RandomFloatInZeroToOne();
00017
                   Math::Vector3f N = payload.normal;
00018
00019
                   float a = m_Material->roughness * m_Material->roughness;
00020
00021
                   float phi = Math::Constants::Tau<float> * Xi.x;
                   float cosTheta = Math::Sqrt((1.f - Xi.y) / (1.f + (a * a - 1.f) * Xi.y));
float sinTheta = Math::Sqrt(1.f - cosTheta * cosTheta);
00022
00023
00024
00025
                   Math::Vector3f H;
00026
                   H.x = Math::Cos(phi) * sinTheta;
00027
                   H.y = Math::Sin(phi) * sinTheta;
                   H.z = cosTheta;
00028
00029
                   Math::Vector3f up = Math::Abs(N.z) < 0.999f ? Math::Vector3f(0.0, 0.0, 1.0) :
00030
      Math::Vector3f(1.0, 0.0, 0.0);
```

4.6 BSDF.cpp 47

```
Math::Vector3f tangent = Math::Normalize(Math::Cross(up, N));
                    Math::Vector3f bitangent = Math::Cross(N, tangent);
00032
00033
                    \label{eq:halfVec} \begin{array}{l} \text{halfVec} = \text{tangent} \ \star \ \text{H.x} \ + \ \text{bitangent} \ \star \ \text{H.y} \ + \ \text{N} \ \star \ \text{H.z}; \\ \text{halfVec} = \ \text{Math::Normalize(halfVec);} \end{array}
00034
00035
00036
                }
00038
                reflectionDirection = Math::Normalize(2.f * Math::Dot(V, halfVec) * halfVec - V);
00039
           }
00040
00041
           auto DistributionGGX = [](const Math::Vector3f &N, const Math::Vector3f &H, float roughness) {
00042
               float a = roughness * roughness;
float a2 = a * a;
00043
00044
                float NdotH = Math::Max(Math::Dot(N, H), 0.f);
00045
                float NdotH2 = NdotH * NdotH;
00046
                float nom = a2; float denom = (NdotH2 * (a2 - 1.f) + 1.f);
00047
00048
                denom = Math::Constants::Pi<float> * denom * denom;
00049
00050
00051
                return nom / denom;
00052
           };
00053
           auto GeometrySchlickGGX = [](float NdotV, float roughness) { float r = (roughness + 1.f); float k = (r * r) / 8.f;
00054
00055
00056
00057
00058
                float nom = NdotV;
                float denom = NdotV * (1.f - k) + k;
00059
00060
00061
                return nom / denom;
00062
           };
00063
00064
           auto GeometrySmith = [&](const Math::Vector3f &N, const Math::Vector3f &V, const Math::Vector3f
      &L, float roughness) {
00065
                float NdotV = Math::Abs(Math::Dot(N, V));
                float NdotL = Math::Abs(Math::Dot(N, L));
float ggx2 = GeometrySchlickGGX(NdotV, roughness);
00066
00067
00068
                float ggx1 = GeometrySchlickGGX(NdotL, roughness);
00069
00070
                return ggx1 * ggx2;
00071
           };
00072
00073
           auto FresnelSchlick = [](float cosTheta, const Math::Vector3f &FO) {
00074
              return F0 + (1.f - F0) * Math::Pow(1.f - cosTheta, 5.f);
00075
00076
00077
           auto SpecularBRDF = [](float D, float G, const Math::Vector3f &F, const Math::Vector3f &V, const
     Math::Vector3f &L, const Math::Vector3f &N) {
    float NdotL = Math::Abs(Math::Dot(N, L));
00078
                float NdotV = Math::Abs(Math::Dot(N, V));
00079
00080
00081
                Math::Vector3f nominator = D \star G \star F;
                float denominator = 4.f * NdotV * NdotL + 0.001f;
00082
00083
                Math:: Vector3f specularBrdf = nominator / denominator;
00084
00085
                return specularBrdf;
00086
           };
00087
00088
           auto DiffuseBRDF = [](const Math::Vector3f &albedo) {
00089
                return albedo * Math::Constants::InversePi<float>;
00090
           };
00091
00092
           auto ImportanceSampleGGXPDF = [](float NDF, float NdotH, float VdotH) {
00093
                return NDF * NdotH / (4.f * VdotH);
00094
00095
00096
           auto CosineSamplingPDF = [](float NdotL) {
00097
               return NdotL * Math::Constants::InversePi<float>;
00098
00099
00100
           Math::Vector3f L = reflectionDirection;
           Math::Vector3f H = Math::Normalize(V + L);
00101
00102
00103
           float NdotL = Math::Abs(Math::Dot(payload.normal, L));
           float NdotH = Math::Abs(Math::Dot(payload.normal, H));
00104
00105
           float VdotH = Math::Abs(Math::Dot(V, H));
00106
00107
           float NdotV = Math::Abs(Math::Dot(payload.normal, V));
00108
           Math:: Vector3f F0 = Math:: Vector3f(0.08f, 0.08f, 0.08f);
00109
00110
           F0 = Math::Lerp(F0 * m_Material->specular, m_Material->albedo, m_Material->metallic);
00111
00112
           float NDF = DistributionGGX(payload.normal, H, m_Material->roughness);
00113
           float G = GeometrySmith(payload.normal, V, L, m_Material->roughness);
00114
           Math::Vector3f F = FresnelSchlick(Math::Max(Math::Dot(H, V), 0.f), F0);
00115
```

```
00116
          Math::Vector3f kS = F;
00117
          Math::Vector3f kD = 1.f - kS;
00118
          kD *= 1.0 - m_Material->metallic;
00119
00120
          Math:: Vector3f specularBrdf = SpecularBRDF (NDF, G, F, V, L, payload.normal);
00121
00122
          float specularPdf = ImportanceSampleGGXPDF(NDF, NdotH, VdotH);
00123
00124
          Math::Vector3f diffuseBrdf = DiffuseBRDF(m_Material->albedo);
00125
          float diffusePdf = CosineSamplingPDF(NdotL);
00126
          Math::Vector3f totalBrdf = (diffuseBrdf * kD + specularBrdf) * NdotL;
00127
00128
          float totalPdf = diffuseRatio * diffusePdf + specularRatio * specularPdf;
00129
00130
          if (totalPdf > Math::Constants::Epsilon<float>) {
00131
             throughput *= totalBrdf / totalPdf;
00132
00133
00134
          return reflectionDirection;
00135 }
```

#### 4.7 BSDF.h

```
00001 #ifndef _BSDF_H
00002 #define _BSDF_H
00004 #include "Material.h"
00005 #include "Ray.h"
00006 #include "HitPayload.h"
00007 #include "math/Math.h"
00008
00009 class BSDF {
00010 public:
00011
         constexpr BSDF(const Material *material) noexcept :
00012
              m_Material(material) {}
00013
00014
          Math:: Vector3f Sample (const Ray &ray, const HitPayload &payload, Math:: Vector3f &throughput)
     noexcept;
00015
00016 private:
00017
          const Material *m_Material;
00018 };
00019
00020 #endif
```

#### 4.8 BVHNode.h

```
00001 #ifndef _BVHNODE_H
00002 #define _BVHNODE_H
00003
00004 #include "Ray.h"
00005 #include "AABB.h"
00006 #include "HitPayload.h"
00007 #include "hittable/HittableObject.h"
8,000
00009 #include <span>
00010 #include <vector>
00011 #include <functional>
00012
00013 struct BVHNode {
00014
        AABB aabb = AABB::Empty();
          BVHNode *left = nullptr, *right = nullptr; const HittableObject *object = nullptr;
00015
00016
00017
00018
          constexpr BVHNode() noexcept = default;
00019
00020
          constexpr BVHNode (BVHNode *left, BVHNode *right) noexcept :
00021
              left(left), right(right), aabb(left->aabb, right->aabb) {}
00022
00023
          constexpr BVHNode(const HittableObject *object) noexcept :
00024
              object(object), aabb(object->GetBoundingBox()) {}
00025
00026
           constexpr bool IsTerminating() const noexcept {
00027
               return object != nullptr;
00028
00029
00030
           inline bool Hit(const Ray &ray, float tMin, float tMax, HitPayload &payload) noexcept {
            if (!aabb.IntersectsRay(ray, tMin, tMax)) {
00032
                    return false;
```

4.8 BVHNode.h

```
}
00034
00035
              if (IsTerminating()) {
00036
                  return object->Hit(ray, tMin, tMax, payload);
00037
00038
              bool anyHit = left->Hit(ray, tMin, tMax, payload);
00040
              anyHit |= right->Hit(ray, tMin, Math::Min(tMax, payload.t), payload);
00041
00042
              return anyHit;
00043
         }
00044
00045
         inline static BVHNode* MakeHierarchySAH(std::span<HittableObjectPtr> objects, int low, int high)
     noexcept {
00046
             if (low + 1 == high) {
00047
                  return new BVHNode(objects[low]);
00048
00049
00050
              int n = high - low;
00051
              std::vector<AABB> pref(n + 1);
00052
              std::vector<AABB> suff(n + 1);
00053
00054
              float minValue = Math::Constants::Infinity<float>;
00055
              int mid = -1:
00056
              int dimension = -1;
00057
00058
              for (int d = 0; d < 3; ++d) {
00059
                  std::sort(objects.begin() + low, objects.begin() + high, c_ComparatorsSAH[d]);
00060
00061
                  pref[0] = AABB::Empty();
                  for (int i = 0; i < n; ++i) {
00062
00063
                      pref[i + 1] = AABB(pref[i], objects[i + low]->GetBoundingBox());
00064
00065
                  suff[n] = AABB::Empty();
for (int i = n - 1; i >= 0; --i) {
00066
00067
                      suff[i] = AABB(objects[i + low]->GetBoundingBox(), suff[i + 1]);
00068
00069
00070
00071
                   float minValueAlongAxis = Math::Constants::Infinity<float>;
                  int index = -1;
for (int i = 0; i < n; ++i) {
00072
00073
     float value = pref[i + 1].GetSurfaceArea() * (float)(i + 1) + suff[i + 1].GetSurfaceArea() * (float)(n - i - 1);
00074
00075
                      if (value < minValueAlongAxis) {</pre>
00076
                          minValueAlongAxis = value;
00077
                           index = i + low;
00078
00079
                  }
08000
00081
                   if (minValueAlongAxis < minValue) {</pre>
00082
                      minValue = minValueAlongAxis;
00083
                      mid = index + 1;
00084
                      dimension = d;
00085
00086
              }
00087
00088
              std::sort(objects.begin() + low, objects.begin() + high, c_ComparatorsSAH[dimension]);
00089
00090
              BVHNode *left = MakeHierarchySAH(objects, low, mid);
00091
              BVHNode *right = MakeHierarchySAH(objects, mid, high);
00092
00093
              return new BVHNode(left, right);
00094
         }
00095
00096
          inline static const std::function<bool(HittableObjectPtr, HittableObjectPtr)> c_ComparatorsSAH[3]
00097
              [](HittableObjectPtr a, HittableObjectPtr b){
00098
                  return a->GetCentroid().x < b->GetCentroid().x;
00100
              [](HittableObjectPtr a, HittableObjectPtr b){
00101
                  return a->GetCentroid().y < b->GetCentroid().y;
00102
              [](HittableObjectPtr a, HittableObjectPtr b){
00103
00104
                  return a->GetCentroid().z < b->GetCentroid().z;
00105
00106
         };
00107
00108
         inline static BVHNode* MakeHierarchyNaive(std::span<HittableObjectPtr> objects, int low, int high)
     noexcept {
00109
             AABB aabb = AABB::Empty();
00110
              for (int i = low; i < high; ++i) {</pre>
00111
                  aabb = AABB(aabb, objects[i]->GetBoundingBox());
00112
00113
00114
              int longestAxisIndex = 0;
00115
              float longestAxisLength = aabb.max.x - aabb.min.x;
```

```
if (aabb.max.y - aabb.min.y > longestAxisLength) {
00117
00118
                  longestAxisIndex = 1;
                  longestAxisLength = aabb.max.y - aabb.min.y;
00119
00120
00121
00122
              if (aabb.max.z - aabb.min.z > longestAxisLength) {
00123
                  longestAxisIndex = 2;
00124
                  longestAxisLength = aabb.max.z - aabb.min.z;
00125
00126
00127
              auto comparator = c_ComparatorsNaive[longestAxisIndex];
00128
00129
              if (low + 1 == high) {
00130
                  return new BVHNode(objects[low]);
00131
00132
00133
              std::sort(objects.begin() + low, objects.begin() + high, comparator);
00134
00135
              int mid = (low + high) / 2;
00136
              BVHNode *left = MakeHierarchyNaive(objects, low, mid);
00137
              BVHNode *right = MakeHierarchyNaive(objects, mid, high);
00138
              return new BVHNode(left, right);
00139
00140
         }
00141
00142
         inline static const std::function<bool(HittableObjectPtr, HittableObjectPtr)>
     c_ComparatorsNaive[3] = {
00143
              [](HittableObjectPtr a, HittableObjectPtr b){
                  return a->GetBoundingBox().min.x < b->GetBoundingBox().min.x;
00144
00145
00146
              [](HittableObjectPtr a, HittableObjectPtr b){
00147
                 return a->GetBoundingBox().min.y < b->GetBoundingBox().min.y;
00148
00149
              [](HittableObjectPtr a, HittableObjectPtr b){
00150
                  return a->GetBoundingBox().min.z < b->GetBoundingBox().min.z;
00151
              }
00152
         };
00153
00154
          constexpr static void FreeMemory(BVHNode *node) noexcept {
00155
             if (node == nullptr) {
                  return:
00156
00157
00158
              FreeMemory(node->left);
00159
00160
              FreeMemory (node->right);
00161
00162
              delete node;
00163
         }
00164 };
00165
00166 #endif
```

#### 4.9 Camera.cpp

```
00001 #include "Camera.h"
00002 #include "Utilities.hpp"
00003
00004 Camera::Camera(int viewportWidth, int viewportHeight, const Math::Vector3f &position, const
                 Math::Vector3f &target, float verticalFovInDegrees, const Math::Vector3f &up) noexcept :
00005
                            \verb|m_ViewportWidth| (\verb|viewportWidth|), \verb|m_ViewportHeight| (\verb|viewportHeight|), \verb|m_Position|), \verb|m_ViewportHeight| (|viewportHeight|), \verb|m_Position|), \verb|m_Position|), \verb|m_ViewportHeight| (|viewportHeight|), \verb|m_Position|), \|m_Position|), \|m_Position
                 \verb|m_Target(target)|, \verb|m_VerticalFovInDegrees(verticalFovInDegrees)|, \verb|m_Up(up)| | \\
00006
                            m_RayDirections.resize(m_ViewportWidth * m_ViewportHeight);
00007
                             ComputeRayDirections();
00008 }
00009
00010 void Camera::OnViewportResize(int viewportWidth, int viewportHeight) noexcept {
00011
                           if (m_ViewportWidth == viewportWidth && m_ViewportHeight == viewportHeight) {
00012
                                        return:
00013
00014
00015
                             m_ViewportWidth = viewportWidth;
00016
                            m_ViewportHeight = viewportHeight;
00017
                            m_RayDirections.resize(m_ViewportWidth * m_ViewportHeight);
00018
00019 }
00020
00021 void Camera::ComputeRayDirections() noexcept {
00022
                             float verticalFovInRadians = Math::ToRadians(m_VerticalFovInDegrees);
00023
00024
                             Math:: Vector3f forward = m_Target - m_Position;
00025
                             float focalLength = Math::Length(forward);
                             float viewportWorldHeight = 2.f * Math::Tan(verticalFovInRadians * 0.5f) * focalLength;
```

4.10 Camera.h 51

```
00027
           float viewportWorldWidth = (viewportWorldHeight * m_ViewportWidth) / m_ViewportHeight;
00028
00029
           Math::Vector3f w = Math::Normalize(forward);
           Math::Vector3f u = Math::Normalize(Math::Cross(w, m_Up));
Math::Vector3f v = Math::Normalize(Math::Cross(w, u));
00030
00031
00032
           Math::Vector3f horizontal = u * viewportWorldWidth;
00034
           Math::Vector3f vertical = v * viewportWorldHeight;
00035
           Math::Vector3f leftUpper = forward - horizontal * 0.5f - vertical * 0.5f;
00036
00037
           for (int i = 0; i < m_ViewportHeight; ++i) {</pre>
               for (int j = 0; j < m_ViewportWidth; ++j) {
    float uScale = (float)(j + Utilities::RandomFloatInNegativeHalfToHalf()) /</pre>
00038
00039
       (m_ViewportWidth - 1);
00040
                    float vScale = (float)(i + Utilities::RandomFloatInNegativeHalfToHalf()) /
       (m_ViewportHeight - 1);
00041
                    m_RayDirections[m_ViewportWidth * i + j] = Math::Normalize(leftUpper + horizontal * uScale
       + vertical * vScale);
00042
00043
00044 }
```

#### 4.10 Camera.h

```
00001 #ifndef _CAMERA_H
00002 #define _CAMERA_H
00003
00004 #include "math/Math.h"
00005
00006 #include <vector>
00007 #include <span>
00008
00009 class Camera {
00010 public:
00011
          inline Camera() noexcept = default;
00012
      Camera(int viewportWidth, int viewportHeight, const Math::Vector3f &position = {0.f, 0.f, 0.f},
const Math::Vector3f &target = {0.f, 0.f, -1.f}, float verticalFovInDegrees = 20.f, const
00013
      Math::Vector3f &up = \{0.f, 1.f, 0.f\}) noexcept;
00014
00015
           ~Camera() noexcept = default;
00016
00017
           void OnViewportResize(int viewportWidth, int viewportHeight) noexcept;
00018
00019
           void ComputeRayDirections() noexcept;
00020
00021
           constexpr std::span<const Math::Vector3f> GetRayDirections() const noexcept {
00022
               return m_RayDirections;
00023
           }
00024
00025
           constexpr Math::Vector3f GetPosition() const noexcept {
00026
             return m_Position;
00027
00028
00029
           constexpr Math::Vector3f& Position() noexcept {
             return m_Position;
00030
00031
00032
00033
           constexpr Math::Vector3f GetTarget() const noexcept {
00034
             return m_Target;
00035
           }
00036
00037
           constexpr Math::Vector3f& Target() noexcept {
00038
             return m_Target;
00039
00040
00041
           constexpr Math::Vector3f GetUp() const noexcept {
          return m_Up;
}
00042
00043
00044
00045
           constexpr Math::Vector3f& Up() noexcept {
          return m_Up;
00046
00047
00048
           constexpr float GetVerticalFovInDegrees() const noexcept {
00049
00050
              return m VerticalFovInDegrees;
00051
00052
00053
           constexpr float& VerticalFovInDegrees() noexcept {
00054
               return m_VerticalFovInDegrees;
00055
00056
00057 private:
```

```
00058     Math::Vector3f m_Position;
00059     Math::Vector3f m_Target;
00060     Math::Vector3f m_Up;
00061     float m_VerticalFovInDegrees;
00062     int m_ViewportWidth, m_ViewportHeight;
00063
00064     std::vector<Math::Vector3f> m_RayDirections;
00065 };
00066
00067 #endif
```

## 4.11 Entrypoint.cpp

```
00001 #include "Application.h"
00002
00003 #include <iostream>
00004
00005 int main(int argc, char **argv)
          int width = 1280, height = 720;
00007
           if (argc > 1 && argc != 3) {
00008
               std::cerr « "To specify initial window parametes use: [width] [height] \n";
00009
               return -1;
           } else if (argc == 3) {
  width = atoi(argv[0]);
  height = atoi(argv[1]);
00010
00011
00012
00013
00014
00015
           Application application (width, height);
00016
           return application.Run();
00017
00018 }
```

## 4.12 HitPayload.h

```
00001 #ifndef _HIT_PAYLOAD_H
00002 #define _HIT_PAYLOAD_H
00003
00004 #include "math/Math.h"
00005 #include "Material.h"
00006
00007 struct HitPayload {
00008     float t;
00009     Math::Vector3f normal;
00010     const Material *material;
00011 };
00012
00013 #endif
```

## 4.13 Image.cpp

```
00001 #include "Image.h"
00002 #include <gl/gl.h>
00003
00004 Image::Image(int width, int height, const uint32_t *data) noexcept :
          m_Width(width), m_Height(height) {
00005
00006
           glGenTextures(1, &m_Descriptor);
00007
           glBindTexture(GL_TEXTURE_2D, m_Descriptor);
00008
          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00009
00010
00011
00012
           glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, m_Width, m_Height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
      (void*)data);
00013
00014
           glBindTexture(GL_TEXTURE_2D, 0);
00015 }
00016
00017 Image::~Image() noexcept {
00018
          glDeleteTextures(1, &m_Descriptor);
00019 }
00020
00021 void Image::UpdateData(const uint32 t *data) noexcept {
           glBindTexture(GL_TEXTURE_2D, m_Descriptor);
00022
00023
00024
           glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

4.14 Image.h 53

### 4.14 Image.h

```
00001 #ifndef _IMAGE_H
00002 #define _IMAGE_H
00003
00004 #include <vector>
00005 #include <cstdint>
00006
00007 class Image {
00008 public:
00009
          Image() = delete;
00010
00011
          Image(int width, int height, const uint32_t *data) noexcept;
00012
00013
          ~Image() noexcept;
00014
00015
          void UpdateData(const uint32 t *data) noexcept;
00016
00017
          constexpr unsigned int GetDescriptor() const noexcept {
00018
            return m_Descriptor;
00019
00020 private:
00021
          int m Width, m Height:
00022
          unsigned int m_Descriptor;
00023 };
00024
00025 #endif
```

## 4.15 Light.h

```
00001 #ifndef _LIGHT_H
00002 #define _LIGHT_H
00003
00004 #include "hittable/HittableObject.h"
00005 #include "math/Math.h"
00006 #include "Ray.h"
00007 #include "HitPayload.h"
80000
00009 class Light {
00010 public:
00011
          constexpr Light(const HittableObjectPtr object, const Math::Vector3f &emission) noexcept :
00012
               m_Object(object), m_Emission(emission) {}
00013
00014
           const HittableObject* GetObject() const noexcept {
00015
             return m_Object;
00016
00017
     Math::Vector3f Sample(const Ray &lightRay, const HitPayload &objectHitPayload, const HitPayload &lightHitPayload, float distance, float distanceSquared) const noexcept {
00018
00019
               constexpr float distanceEpsilon = 0.01f;
               if (Math::Abs(lightHitPayload.t - distance) > distanceEpsilon) {
00020
00021
                   return Math::Vector3f(0.f);
00022
00023
               float pdf = distanceSquared / (Math::Dot(lightHitPayload.normal, -lightRay.direction) *
00024
      m_Object->GetArea());
00025
00026
               constexpr float pdfEpsilon = 0.01f;
00027
               if (pdf <= pdfEpsilon) {</pre>
00028
                    return Math::Vector3f(0.f);
00029
00030
               Math::Vector3f brdf = objectHitPayload.material->albedo * Math::Constants::InversePi<float> *
00031
      Math::Dot(objectHitPayload.normal, lightRay.direction) * m_Emission;
00032
00033
               return brdf / pdf;
00034
           }
00035
00036 private:
          const HittableObject* m_Object;
00038
           Math::Vector3f m_Emission;
```

```
00039 };
00040
00041 #endif
```

#### 4.16 Material.h

```
00001 #ifndef _MATERIAL_H
00002 #define _MATERIAL_H
00003
00004 #include "math/Math.h"
00005
00006 struct Material {
00007
          Math::Vector3f albedo;
80000
          float metallic;
00009
           float specular;
00010
          float roughness;
00011
          float emissionPower;
00012
00013
          int index;
00014
00015
          constexpr Math::Vector3f GetEmission() const noexcept {
00016
              return albedo * emissionPower;
00017
00018 };
00019
00020 #endif
```

## 4.17 Ray.h

## 4.18 Renderer.cpp

```
00001 #include "Renderer.h"
00002 #include "Utilities.hpp"
00003 #include "BSDF.h"
00004
00005 #define STB_IMAGE_WRITE_IMPLEMENTATION 00006 #include "../stb-master/stb_image_write.h"
00008 #include <vector>
00009 #include <thread>
00010 #include <cstring>
00011
00012 Renderer::Renderer(int width, int height) noexcept :
00013
         m_Width(width), m_Height(height) {
          m_ImageData = new uint32_t[m_Width * m_Height];
m_Image = new Image(m_Width, m_Height, m_ImageData);
00015
00016
           m_AccumulationData = new Math::Vector4f[m_Width * m_Height];
00017
00018
           m AvailableThreads = std::thread::hardware concurrency();
00019
           m_UsedThreads = 1;
           m_LinesPerThread = (m_Height + m_UsedThreads - 1) / m_UsedThreads;
00020
00021 }
00022
00023 Renderer::~Renderer() noexcept {
00024
          if (m_Image != nullptr) {
00025
               delete m_Image;
00026
00027
           if (m_ImageData != nullptr) {
00028
               delete[] m_ImageData;
00029
00030
           if (m_AccumulationData != nullptr) {
00031
               delete[] m_AccumulationData;
00032
00033 }
```

4.18 Renderer.cpp 55

```
00035 void Renderer::SaveImage(const char *filename) const noexcept {
00036
          stbi_write_png(filename, m_Width, m_Height, 4, m_ImageData, m_Width * 4);
00037 }
00038
00039 void Renderer::OnResize(int width, int height) noexcept {
          if (m_Width == width && m_Height == height) {
00041
00042
          }
00043
          m_Width = width;
00044
          m_Height = height;
00045
00046
00047
          if (m_ImageData != nullptr) {
00048
              delete m_ImageData;
00049
              m_ImageData = new uint32_t[m_Width * m_Height];
00050
00051
          if (m Image != nullptr) {
00052
              delete m_Image;
00053
              m_Image = new Image(m_Width, m_Height, m_ImageData);
00054
00055
          if (m_AccumulationData != nullptr) {
00056
              delete m_AccumulationData;
00057
              m AccumulationData = new Math::Vector4f[m Width * m Height];
00058
          }
00059 }
00060
00061 void Renderer::Render(const Camera &camera, std::span<const HittableObjectPtr> objects,
      std::span<const Light> lightSources, std::span<const Material> materials) noexcept {
00062
          m_Camera = &camera;
m_Objects = objects;
00063
00064
          m_LightSources = lightSources;
00065
          m_Materials = materials;
00066
00067
          if (!m_Accumulate)
00068
              m_FrameIndex = 1;
00069
          }
00070
00071
          if (m FrameIndex == 1) {
00072
              memset(m_AccumulationData, 0, m_Width * m_Height * sizeof(Math::Vector4f));
00073
00074
00075
          float inverseFrameIndex = 1.f / m FrameIndex:
00076
          float inverseGamma = 1.f / m_Gamma;
00077
00078
          std::vector<std::thread> handles;
00079
          handles.reserve(m_UsedThreads);
00080
00081
          for (int i = 0; i < m_Height; i += m_LinesPerThread) {</pre>
              handles.emplace_back([this, i, inverseFrameIndex, inverseGamma]() {
00082
                   int nextBlock = i + m_LinesPerThread;
00083
00084
                   int limit = Math::Min(nextBlock, m_Height);
00085
                   for (int t = i; t < limit; ++t) {</pre>
                       for (int j = 0; j < m_Width; ++j) {</pre>
00086
                           \label{eq:m_AccumulationData} $$ m_AccumulationData[m_Width * t + j] += PixelProgram(t, j); $$
00087
00088
00089
                           Math::Vector4f color = m_AccumulationData[m_Width * t + j];
00090
00091
                           color *= inverseFrameIndex;
00092
                           color = Utilities::CorrectGamma(color, inverseGamma);
00093
                           color = Math::Clamp(color, 0.f, 1.f);
00094
00095
                           m_ImageData[m_Width * t + j] = Utilities::ConvertColorToRGBA(color);
00096
00097
00098
              });
00099
          }
00100
00101
          for (auto &handle : handles) {
00102
              handle.join();
00103
00104
00105
          m_Image->UpdateData(m_ImageData);
00106
00107
          if (m Accumulate) {
00108
              ++m_FrameIndex;
00109
00110 }
00111
00112 void Renderer::Render(const Camera & camera, const AccelerationStructure & accelerationStructure.
      std::span<const Light> lightSources, std::span<const Material> materials) noexcept {
00113
          m_Camera = &camera;
00114
          m_AccelerationStructure = &accelerationStructure;
00115
          m_LightSources = lightSources;
00116
          m_Materials = materials;
00117
00118
          if (!m Accumulate) {
```

```
00119
              m_FrameIndex = 1;
00120
          }
00121
00122
          if (m FrameIndex == 1) {
               \texttt{memset} \, (\texttt{m\_AccumulationData, 0, m\_Width * m\_Height * sizeof(Math::Vector4f));}
00123
00124
          }
00125
00126
           float inverseFrameIndex = 1.f / m_FrameIndex;
00127
          float inverseGamma = 1.f / m_Gamma;
00128
          std::vector<std::thread> handles;
00129
00130
          handles.reserve(m UsedThreads);
00131
00132
          for (int i = 0; i < m_Height; i += m_LinesPerThread) {</pre>
00133
               handles.emplace_back([this, i, inverseFrameIndex, inverseGamma]() {
00134
                   int nextBlock = i + m_LinesPerThread;
                   int limit = Math::Min(nextBlock, m_Height);
00135
                   for (int t = i; t < limit; ++t) {
    for (int j = 0; j < m_Width; ++j) {</pre>
00136
00137
00138
                            m_AccumulationData[m_Width * t + j] += AcceleratedPixelProgram(t, j);
00139
00140
                            Math::Vector4f color = m_AccumulationData[m_Width * t + j];
00141
                            color *= inverseFrameIndex;
00142
00143
                            color = Utilities::CorrectGamma(color, inverseGamma);
00144
                            color = Math::Clamp(color, 0.f, 1.f);
00145
00146
                            m_ImageData[m_Width * t + j] = Utilities::ConvertColorToRGBA(color);
00147
                       }
00148
                   }
00149
               });
00150
          }
00151
00152
           for (auto &handle : handles) {
00153
               handle.join();
00154
00155
00156
          m_Image->UpdateData(m_ImageData);
00157
00158
          if (m_Accumulate) {
00159
               ++m_FrameIndex;
00160
          }
00161 }
00162
00163 Math::Vector4f Renderer::PixelProgram(int i, int j) const noexcept {
00164
00165
          ray.origin = m_Camera->GetPosition();
00166
          ray.direction = m_Camera->GetRayDirections()[m_Width * i + j];
00167
          Math::Vector3f light(0.f), throughput(1.f);
00168
00169
          for (int i = 0; i < m_RayDepth; ++i)</pre>
00170
              HitPayload payload = TraceRay(ray);
00171
00172
               if (payload.t < 0.f) {</pre>
                   light += throughput * m_OnRayMiss(ray);
00173
00174
                   break;
00175
00176
               const Material *material = payload.material;
auto emission = material->GetEmission();
00177
00178
00179
00180
              light += emission * throughput;
00181
00182
               if (material->emissionPower > 0.f) {
00183
00184
00185
00186
               auto hitPoint = ray.origin + ray.direction * payload.t;
               for (auto lightSource : m_LightSources) {
00187
                   auto pointOnLight =
00188
      lightSource.GetObject() ->SampleUniform({Utilities::RandomFloatInZeroToOne(),
      Utilities::RandomFloatInZeroToOne()});
00189
00190
                   auto toLight = pointOnLight - hitPoint;
                   float distanceSquared = Math::Dot(toLight, toLight);
float distance = Math::Sqrt(distanceSquared);
00191
00192
00193
00194
                   Ray lightRay(hitPoint, toLight / distance);
00195
00196
                   HitPayload lightHitPayload = TraceRay(lightRay);
00197
00198
                   light += throughput * lightSource.Sample(lightRay, payload, lightHitPayload, distance,
      distanceSquared);
00199
00200
               BSDF bsdf (material);
00201
00202
               auto direction = bsdf.Sample(ray, payload, throughput);
```

4.18 Renderer.cpp 57

```
00203
00204
              ray.origin = hitPoint;
00205
              ray.direction = direction;
00206
          }
00207
00208
          return {light.r, light.g, light.b, 1.f};
00210
00211 Math::Vector4f Renderer::AcceleratedPixelProgram(int i, int j) const noexcept {
00212
00213
          ray.origin = m_Camera->GetPosition();
          ray.direction = m_Camera->GetRayDirections()[m_Width * i + j];
00214
00215
00216
          Math:: Vector3f light(0.f), throughput(1.f);
00217
          for (int i = 0; i < m_RayDepth; ++i)</pre>
00218
              HitPayload payload = AcceleratedTraceRay(ray);
00219
00220
              if (payload.t < 0.f) {
                   light += throughput * m_OnRayMiss(ray);
00221
00222
                   break;
00223
00224
              const Material *material = payload.material;
Math::Vector3f emission = material->GetEmission();
00225
00226
00227
00228
              light += emission * throughput;
00229
00230
              if (material->emissionPower > 0.f) {
00231
00232
              }
00233
00234
              auto hitPoint = ray.origin + ray.direction * payload.t;
00235
              for (auto lightSource : m_LightSources) {
00236
                   auto pointOnLight =
      lightSource.GetObject() ->SampleUniform({Utilities::RandomFloatInZeroToOne(),
      Utilities::RandomFloatInZeroToOne() });
00237
00238
                  auto toLight = pointOnLight - hitPoint;
00239
                   float distanceSquared = Math::Dot(toLight, toLight);
00240
                  float distance = Math::Sqrt(distanceSquared);
00241
00242
                  Ray lightRay(hitPoint, toLight / distance);
00243
00244
                  HitPayload lightHitPayload = TraceRay(lightRay);
00245
00246
                  light += throughput * lightSource.Sample(lightRay, payload, lightHitPayload, distance,
     distanceSquared);
00247
             }
00248
              BSDF bsdf (material);
00249
              auto direction = bsdf.Sample(ray, payload, throughput);
00251
00252
              ray.origin += ray.direction * payload.t;
00253
              ray.direction = direction;
00254
00255
00256
          return {light.r, light.g, light.b, 1.f};
00257 }
00258
00259 HitPayload Renderer::TraceRay(const Ray &ray) const noexcept {
00260
          HitPayload payload;
payload.t = Math::Constants::Infinity<float>;
00261
00262
          payload.normal = Math::Vector3f(0.f);
00263
          payload.material = nullptr;
00264
00265
          int objectCount = (int)m_Objects.size();
00266
          for (int i = 0; i < objectCount; ++i) {
    m_Objects[i]->Hit(ray, 0.01f, Math::Min(payload.t, Math::Constants::Infinity<float>),
00267
      payload);
00268
          }
00269
00270
          if (payload.material == nullptr) {
00271
             return Miss(ray);
00272
00273
          payload.normal = Math::Dot(ray.direction, payload.normal) > Math::Constants::Epsilon<float> ?
     -payload.normal : payload.normal;
00275
00276
          return payload;
00277 }
00278
00279 HitPayload Renderer::AcceleratedTraceRay(const Ray &ray) const noexcept {
00280
          HitPayload payload;
00281
          payload.t = Math::Constants::Infinity<float>;
00282
          payload.normal = Math::Vector3f(0.f);
00283
          payload.material = nullptr;
00284
```

```
m_AccelerationStructure->Hit(ray, 0.01f, Math::Constants::Infinity<float>, payload);
00286
00287
          if (payload.material == nullptr) {
00288
             return Miss(ray);
00289
00290
00291
          payload.normal = Math::Dot(ray.direction, payload.normal) > Math::Constants::Epsilon<float> ?
      -payload.normal : payload.normal;
00292
00293
          return payload;
00294 }
00295
00296 HitPayload Renderer::Miss(const Ray &ray) const noexcept {
00297
          HitPayload payload;
00298
          payload.t = -1.f;
00299
          return payload;
00300 3
```

#### 4.19 Renderer.h

```
00001 #ifndef _RENDERER_H
00002 #define _RENDERER_H
00003
00004 #include "Image.h"
00005 #include "Camera.h"
00006 #include "Scene.h"
00007 #include "HitPayload.h"
00008 #include "Ray.h"
00009 #include "math/Math.h"
00010 #include "AccelerationStructure.h"
00011 #include "Material.h"
00012 #include "Light.h"
00014 #include <functional>
00015 #include <span>
00016
00017 class Renderer {
00018 public:
00019
           Renderer() = delete;
00020
00021
           Renderer(int width, int height) noexcept;
00022
00023
           ~Renderer() noexcept;
00024
00025
           using typ_t = HittableObject*;
00026
      void Render(const Camera &camera, std::span<const HittableObjectPtr> objects, std::span<const
Light> lightSources, std::span<const Material> materials) noexcept;
00027
00028
           void Render(const Camera &camera, const AccelerationStructure &accelerationStructure,
00029
      std::span<const Light> lightSources, std::span<const Material> materials) noexcept;
00030
00031
           constexpr bool& Accumulate() noexcept {
00032
              return m_Accumulate;
00033
           }
00034
00035
           constexpr bool& Accelerate() noexcept {
           return m_Accelerate;
00036
00037
00038
00039
           constexpr int GetFrameIndex() const noexcept {
00040
               return m_FrameIndex;
00041
           }
00042
00043
           constexpr Image* GetImage() const noexcept {
00044
             return m_Image;
00045
00046
00047
           void SaveImage(const char *filename) const noexcept;
00048
00049
           void OnResize(int width, int height) noexcept;
00050
00051
           constexpr int GetAvailableThreadCount() const noexcept {
00052
               return m_AvailableThreads;
00053
00054
00055
           constexpr int GetUsedThreadCount() const noexcept {
00056
              return m_UsedThreads;
00057
00058
           constexpr void SetUsedThreadCount(int usedThreads) noexcept {
00059
               m_UsedThreads = Math::Clamp(usedThreads, 1, m_AvailableThreads);
m_LinesPerThread = (m_Height + m_UsedThreads - 1) / m_UsedThreads;
00060
00061
```

4.20 Scene.h 59

```
00062
          }
00063
00064
          constexpr int& UsedThreadCount() noexcept {
00065
             return m_UsedThreads;
00066
00067
          inline void OnRayMiss(std::function<Math::Vector3f(const Ray&)> onRayMiss) noexcept {
00068
00069
              m_OnRayMiss = onRayMiss;
00070
00071
00072
          constexpr int& RayDepth() noexcept {
00073
             return m_RayDepth;
00074
00075
00076
          constexpr float& Gamma() noexcept {
00077
             return m_Gamma;
00078
00079
00080 private:
00081
          Math::Vector4f PixelProgram(int u, int j) const noexcept;
00082
00083
          Math::Vector4f AcceleratedPixelProgram(int i, int j) const noexcept;
00084
00085
          HitPayload TraceRay(const Ray &ray) const noexcept;
00086
00087
          HitPayload AcceleratedTraceRay(const Ray &ray) const noexcept;
00088
00089
          HitPayload Miss (const Ray &ray) const noexcept;
00090
00091 private:
          int m_Width, m_Height;
Image *m_Image = nullptr;
00092
00093
00094
          uint32_t *m_ImageData = nullptr;
00095
00096
          std::function<Math::Vector3f(const Ray&)> m_OnRayMiss = [](const Ray&){ return Math::Vector3f(0.f,
     0.f, 0.f); };
00097
00098
          int m_AvailableThreads;
00099
          int m_UsedThreads;
00100
          int m_LinesPerThread;
00101
00102
          int m RayDepth = 5;
00103
00104
          const Camera *m_Camera = nullptr;
00105
          std::span<const HittableObjectPtr> m_Objects;
00106
          std::span<const Light> m_LightSources;
00107
          const AccelerationStructure *m_AccelerationStructure = nullptr;
00108
          std::span<const Material> m_Materials;
00109
00110
          bool m_Accumulate = false;
00111
          Math::Vector4f *m_AccumulationData = nullptr;
00112
          int m_FrameIndex = 1;
00113
00114
          bool m_Accelerate = false;
00115
00116
          float m Gamma = 2.f;
00117 };
00118
00119 #endif
```

### 4.20 Scene.h

```
00001 #ifndef _SCENE_H
00002 #define _SCENE_H
00003
00004 #include "hittable/Sphere.h"
00005 #include "hittable/Triangle.h"
00006 #include "hittable/Box.h"
00007 #include "hittable/Model.h"
00008 #include "Material.h"
00009 #include "Camera.h"
00010
00011 #include <vector>
00012 #include <fstream>
00013 #include <array>
00014 #include <exception>
00015 #include <optional>
00016
00017 struct Scene {
             std::vector<Shapes::Sphere> spheres;
00018
00019
             std::vector<Shapes::Triangle> triangles;
00020
             std::vector<Shapes::Box> boxes;
            std::vector<Model*> models;
```

```
00022
           std::vector<Material> materials;
00023
00024
           std::optional<std::string> Serialize(std::ostream &os) const noexcept {
00025
               os.exceptions(std::ios::badbit | std::ios::failbit);
00026
00027
00028
00029
                    TrySerialize(os);
00030
                 catch (std::exception &e) {
00031
                    return e.what();
               }
00032
00033
00034
               return {};
00035
00036
00037
           std::optional<std::string> Deserialize(std::istream &is) noexcept {
               is.exceptions(std::ios::eofbit | std::ios::badbit | std::ios::failbit);
00038
00039
00041
                    TryDeserialize(is);
                 catch (std::exception &e) {
00042
00043
                    return e.what();
00044
               }
00045
00046
               return {};
00047
         }
00048
00049 private:
00050
          void TrySerialize(std::ostream &os) const {
00051
               int materialCount = static_cast<int>(materials.size());
               int sphereCount = static_cast<int>(spheres.size());
00052
00053
                int triangleCount = static_cast<int>(triangles.size());
00054
                int boxCount = static_cast<int>(boxes.size());
00055
               int modelCount = static_cast<int>(models.size());
00056
               os.write(reinterpret_cast<const char*>(&materialCount), sizeof(materialCount));
00057
               os.write(reinterpret_cast<const char*>(&sphereCount), sizeof(sphereCount));
os.write(reinterpret_cast<const char*>(&triangleCount), sizeof(triangleCount));
00058
00060
               os.write(reinterpret_cast<const char*>(&boxCount), sizeof(boxCount));
00061
               os.write(reinterpret_cast<const char*>(&modelCount), sizeof(modelCount));
00062
00063
               for (const auto &material : materials) {
00064
                   os.write(reinterpret_cast<const char*>(&material), sizeof(material));
00065
00066
00067
               for (const auto &sphere : spheres) {
00068
                   os.write(reinterpret_cast<const char*>(&sphere.center), sizeof(sphere.center));
                    os.write(reinterpret_cast<const char*>(&sphere.radius), sizeof(sphere.radius)); os.write(reinterpret_cast<const char*>(&sphere.material->index),
00069
00070
      sizeof(sphere.material->index));
00071
00072
00073
                for (const auto &triangle : triangles) {
                   os.write(reinterpret_cast<const char*>(&triangle.vertices), sizeof(triangle.vertices));
os.write(reinterpret_cast<const char*>(&triangle.normal), sizeof(triangle.normal));
os.write(reinterpret_cast<const char*>(&triangle.material->index),
00074
00075
00076
      sizeof(triangle.material->index));
00077
              }
00078
00079
               for (const auto &box : boxes) {
                   os.write(reinterpret_cast<const char*>(&box.min), sizeof(box.min));
os.write(reinterpret_cast<const char*>(&box.max), sizeof(box.max));
08000
00081
00082
                    os.write(reinterpret_cast<const char*>(&box.material->index),
      sizeof(box.material->index));
00083
               }
00084
00085
               for (const auto model : models) {
                    auto pathToFile = model->GetPathToFile().string();
00086
00087
                    int pathToFileLength = static_cast<int>(pathToFile.length());
00088
00089
                    auto materialDirectory = model->GetMaterialDirectory().string();
00090
                    int materialDirectoryLength = static_cast<int>(materialDirectory.length());
00091
00092
                    os.write(reinterpret_cast<const char*>(&pathToFileLength), sizeof(pathToFileLength));
00093
                    os.write(pathToFile.data(), pathToFileLength);
00094
00095
                    os.write(reinterpret_cast<const char*>(&materialDirectoryLength),
      sizeof(materialDirectoryLength));
00096
                    os.write(materialDirectory.data(), materialDirectoryLength);
00097
               }
00098
00099
               // todo! loop for models here
00100
00101
               auto position = camera.GetPosition();
00102
               auto target = camera.GetTarget();
               auto verticalFovInDegrees = camera.GetVerticalFovInDegrees();
00103
00104
               auto up = camera.GetUp();
```

4.20 Scene.h 61

```
00105
00106
               os.write(reinterpret_cast<const char*>(&position), sizeof(position));
               os.write(reinterpret_cast<const char*>(&target), sizeof(target));
os.write(reinterpret_cast<const char*>(&verticalFovInDegrees), sizeof(verticalFovInDegrees));
00107
00108
00109
               os.write(reinterpret_cast<const char*>(&up), sizeof(up));
00110
          }
00111
00112
          void TryDeserialize(std::istream &is) {
00113
              int materialCount;
00114
               int sphereCount;
00115
               int triangleCount;
00116
               int boxCount:
00117
               int modelCount;
00118
               is.read(reinterpret_cast<char*>(&materialCount), sizeof(materialCount));
00119
00120
               is.read(reinterpret_cast<char*>(&sphereCount), sizeof(sphereCount));
               is.read(reinterpret_cast<char*>(&triangleCount), sizeof(triangleCount));
00121
               is.read(reinterpret_cast<char*>(&boxCount), sizeof(boxCount));
is.read(reinterpret_cast<char*>(&modelCount), sizeof(modelCount));
00122
00123
00124
00125
               materials.clear();
00126
               materials.resize(materialCount);
00127
               for (auto &material : materials) {
00128
                   is.read(reinterpret cast<char*>(&material), sizeof(material));
00129
00130
               materials.shrink_to_fit();
00131
00132
               spheres.clear();
00133
               spheres.reserve(sphereCount);
00134
               while (sphereCount--) {
00135
                   Math:: Vector3f center:
00136
                   float radius;
00137
                    int materialIndex;
                   is.read(reinterpret_cast<char*>(&center), sizeof(center));
00138
00139
                   is.read(reinterpret_cast<char*>(&radius), sizeof(radius));
                   is.read(reinterpret_cast<char*>(&materialIndex), sizeof(materialIndex));
00140
00141
00142
                   spheres.emplace_back(center, radius, &materials[materialIndex]);
00143
00144
               spheres.shrink_to_fit();
00145
00146
               triangles.clear();
00147
               triangles.reserve(triangleCount);
00148
               while (triangleCount--) {
                   Math::Vector3f vertices[3];
00149
00150
                   Math::Vector3f normal;
00151
                   int materialIndex:
00152
                   is.read(reinterpret_cast<char*>(&vertices), sizeof(vertices));
                   is.read(reinterpret_cast<char*>(&normal), sizeof(normal));
is.read(reinterpret_cast<char*>(&materialIndex), sizeof(materialIndex));
00153
00154
00155
                   triangles.emplace_back(std::array<Math::Vector3f, 3>{vertices[0], vertices[1],
00156
      vertices[2]}, normal, &materials[materialIndex]);
00157
               triangles.shrink_to_fit();
00158
00159
00160
               boxes.clear();
00161
               boxes.reserve(boxCount);
00162
               while (boxCount--) {
00163
                   Math:: Vector3f min, max;
00164
                   int materialIndex;
                   is.read(reinterpret_cast<char*>(&min), sizeof(min));
00165
00166
                   is.read(reinterpret_cast<char*>(&max), sizeof(max));
                   is.read(reinterpret_cast<char*>(&materialIndex), sizeof(materialIndex));
00167
00168
00169
                   boxes.emplace_back(min, max, &materials[materialIndex]);
00170
00171
               boxes.shrink to fit():
00172
00173
00174
               for (const auto model : models) {
00175
                   delete model;
00176
               }
00177
00178
               models.clear();
00179
               models.reserve(modelCount);
00180
               while (modelCount--) {
00181
                   int pathToFileLength;
00182
                   is.read(reinterpret_cast<char*>(&pathToFileLength), sizeof(pathToFileLength));
00183
00184
                   std::vector<char> buffer(pathToFileLength);
00185
                   is.read(buffer.data(), pathToFileLength);
00186
00187
                   std::string pathToFile(buffer.data(), pathToFileLength);
00188
00189
                   int materialDirectoryLength;
00190
                   is.read(reinterpret cast<char*>(&materialDirectoryLength),
```

```
sizeof(materialDirectoryLength));
00191
00192
                   buffer.resize(materialDirectoryLength);
00193
                   is.read(buffer.data(), materialDirectoryLength);
00194
00195
                   std::string materialDirectory(buffer.data(), materialDirectoryLength);
00196
00197
                   auto result = Model::LoadOBJ(pathToFile, materialDirectory);
00198
                   if (result.IsFailure()) {
                        continue;
00199
00200
                   }
00201
00202
                   models.push_back(result.model);
00203
00204
               models.shrink_to_fit();
00205
               \verb|is.read(reinterpret_cast<char*>(\&camera.Position()), \verb|sizeof(camera.Position())|); \\
00206
               is.read(reinterpret_cast<char*>(&camera.Target()));
is.read(reinterpret_cast<char*>(&camera.VerticalFovInDegrees()),
00207
00208
      sizeof(camera.VerticalFovInDegrees()));
00209
               is.read(reinterpret_cast<char*>(&camera.Up()), sizeof(camera.Up()));
00210
00211 };
00212
00213 #endif
```

#### 4.21 Timer.h

```
00001 #ifndef _TIMER_H
00002 #define _TIMER_H
00003
00004 #include <chrono>
00005 #include <functional>
00006
00007 class Timer {
00008 public:
00009
         constexpr Timer() noexcept = delete;
00010
          constexpr Timer(const Timer&) = delete;
00011
          constexpr Timer(Timer&&) = delete;
00012
00013
          inline static double MeasureInMillis(std::function<void()> f) {
00014
              auto t1 = std::chrono::high_resolution_clock::now();
00015
              f();
00016
              auto t2 = std::chrono::high resolution clock::now();
              static_cast<std::chrono::duration<float, std::milli>>(t2 - t1).count();
00018
              return std::chrono::duration_cast<std::chrono::duration<float, std::milli»(t2 - t1).count();</pre>
00019
00020 };
00021
00022 #endif
```

## 4.22 Utilities.hpp

```
00001 #ifndef _UTILITIES_HPP
00002 #define _UTILITIES_HPP
00003
00004 #include <chrono>
00005 #include <random>
00007 #include "math/Math.h"
00008
00009 namespace Utilities {
          inline static uint32_t s_RandomEngineState =
00010
     std::chrono::high_resolution_clock::now().time_since_epoch().count();
          thread_local inline static std::mt19937_64
      s_RandomNumberGenerator(std::chrono::high_resolution_clock::now().time_since_epoch().count());
00012
          inline static std::uniform_real_distribution<> s_ZeroToOne(0.f, 1.f);
00013
00014
          inline uint32_t RandomUint() {
00015
              uint32 t state = s RandomEngineState;
00016
              s_RandomEngineState = state * 747796405u + 2891336453u;
00017
              uint32_t word = ((state » ((state » 28u) + 4u)) ^ state) * 277803737u;
00018
              return (word » 22u) ^ word;
00019
          }
00020
00021
          inline float RandomFloatInZeroToOne() noexcept {
00022
              return s_ZeroToOne(s_RandomNumberGenerator);
00024
```

4.22 Utilities.hpp 63

```
inline float RandomFloatInNegativeHalfToHalf() noexcept {
00026
             return RandomFloatInZeroToOne() - 0.5f;
00027
          }
00028
          inline float RandomFloatInNegativeToOne() noexcept {
00029
          return RandomFloatInZeroToOne() * 2.f - 1.f;
}
00030
00032
00033
          inline int RandomIntInRange(int min, int max) noexcept {
00034
              return std::uniform_int_distribution<> (min, max) (s_RandomNumberGenerator);
00035
00036
00037
          inline Math::Vector3f RandomInUnitSphere() noexcept {
00038
             while (true) {
00039
                 Math::Vector3f result(RandomFloatInNegativeToOne(), RandomFloatInNegativeToOne(),
     RandomFloatInNegativeToOne());
00040
                  if (Math::Dot(result, result) < 1.f) {</pre>
00041
                       return result;
00042
                  }
00043
              }
00044
          }
00045
00046
          inline Math:: Vector3f RandomUnitVector() noexcept {
00047
              return Math::Normalize(RandomInUnitSphere());
00048
00049
00050
          inline Math::Vector3f RandomInHemisphere(const Math::Vector3f &normal) {
00051
              Math::Vector3f randomUnitVector = RandomUnitVector();
00052
              if (Math::Dot(randomUnitVector, normal) < 0.f) {</pre>
00053
                   return -randomUnitVector;
00054
              }
00055
00056
              return randomUnitVector;
00057
          }
00058
          constexpr float InverseSqrtFast(float x) {
00059
00060
              constexpr float threeHalfs = 1.5f;
00061
00062
              float halfX = x * 0.5f;
00063
              uint32_t i = *(uint32_t*)&x;
00064
              i = 0x5f3759df - (i \gg 1);
00065
00066
              x = *(float*)&i:
00067
              x = x * (threeHalfs - halfX * x * x);
00068
00069
00070
          }
00071
00072
          inline Math:: Vector3f RandomUnitVectorFast() noexcept {
00073
             Math::Vector3f v = RandomInUnitSphere();
00074
              return v * InverseSqrtFast(Math::Dot(v, v));
00075
00076
00077
          inline Math::Vector3f RandomInHemisphereFast(const Math::Vector3f &normal) {
00078
              Math::Vector3f randomUnitVector = RandomUnitVectorFast();
00079
              if (Math::Dot(randomUnitVector, normal) < 0.f) {</pre>
00080
                   return -randomUnitVector;
00081
00082
00083
              return randomUnitVector;
00084
          }
00085
00086
          inline Math::Vector3f RandomCosineDirection() {
             float r1 = RandomFloatInZeroToOne();
float r2 = RandomFloatInZeroToOne();
00087
00088
00089
              constexpr float twoPi = 2.f * std::numbers::pi;
00090
00091
00092
              float phi = twoPi * r1;
               float x = cos(phi) * sqrt(r2);
00093
00094
               float y = \sin(phi) * sqrt(r2);
              float z = sqrt(1.f - r2);
00095
00096
00097
              return {x, y, z};
00098
          }
00099
00100
          constexpr uint32_t AsUint(float x) {
00101
            return *(uint32_t*)&x;
00102
          }
00103
          constexpr float AsFloat(uint32 t x) {
00104
00105
             return *(float*)&x;
00106
00107
          constexpr float PowFast(float x, float exp) {
   constexpr float oneAsUint = 0x3f800000u;
00108
00109
              return AsFloat(int(exp * (AsUint(x) - oneAsUint)) + oneAsUint);
00110
```

```
00111
           }
00112
           constexpr bool AlmostZero(const Math::Vector3f &v) {
00113
               constexpr float epsilon = std::numeric_limits<float>::epsilon();
00114
00115
               00116
00117
00118
           constexpr uint32_t ConvertColorToRGBA(const Math::Vector4f &color) noexcept {
             uint8_t r = (uint8_t) (color.r * 255.0f);
uint8_t g = (uint8_t) (color.g * 255.0f);
uint8_t b = (uint8_t) (color.b * 255.0f);
uint8_t a = (uint8_t) (color.a * 255.0f);
00119
00120
00121
00122
00123
00124
               return (a « 24) | (b « 16) | (g « 8) | r;
00125
00126
00127
           constexpr Math::Vector4f CorrectGamma(const Math::Vector4f &color, float inverseGamma) {
00128
               return {
00129
                   Math::Pow(color.r, inverseGamma),
                    Math::Pow(color.g, inverseGamma),
Math::Pow(color.b, inverseGamma),
00130
00131
00132
                    color.a
00133
               } ;
00134
          }
00135
00136
           constexpr Math::Vector4f CorrectGammaFast(const Math::Vector4f &color, float inverseGamma) {
00137
00138
                    PowFast(color.r, inverseGamma),
                    PowFast(color.g, inverseGamma),
PowFast(color.b, inverseGamma),
00139
00140
00141
                    color.a
00142
               };
00143
00144 }
00145
00146 #endif
```

# Index

$\sim$ AccelerationStructure	c_ComparatorsNaive
AccelerationStructure, 7	BVHNode, 16
$\sim$ Image	c_ComparatorsSAH
Image, 22	BVHNode, 17
$\sim$ Renderer	Camera, 18
Renderer, 27	Camera, 18
	ComputeRayDirections, 19
AABB, 5	GetPosition, 19
AABB, 5	GetRayDirections, 19
Empty, 6	GetTarget, 19
GetSurfaceArea, 6	GetUp, 19
IntersectsRay, 6	GetVerticalFovInDegrees, 20
max, 7	OnViewportResize, 20
min, 7	Position, 20
aabb	Target, 20
BVHNode, 16	Up, 20
Accelerate	VerticalFovInDegrees, 21
Renderer, 28	camera
AccelerationStructure, 7	Scene, 33
$\sim$ AccelerationStructure, 7	ComputeRayDirections
AccelerationStructure, 7	Camera, 19
Hit, 8	
Update, 8	Deserialize
Accumulate	Scene, 33
Renderer, 28	direction
albedo	Ray, 26
Material, 25	
Application, 8	emissionPower
Application, 9	Material, 25
Run, 10	Empty
	AABB, 6
boxes	
Scene, 33	FreeMemory
BSDF, 10	BVHNode, 14
BSDF, 11	
Sample, 11	Gamma
BVHNode, 13	Renderer, 28
aabb, 16	GetAvailableThreadCount
BVHNode, 14	Renderer, 28
c_ComparatorsNaive, 16	GetDescriptor
c_ComparatorsSAH, 17	Image, 23
FreeMemory, 14	GetEmission
Hit, 14	Material, 25
IsTerminating, 14	GetFrameIndex
left, 17	Renderer, 28
MakeHierarchyNaive, 15	GetImage
MakeHierarchySAH, 15	Renderer, 29
object, 17	GetObject
right, 17	Light, 24
<del>y                                    </del>	GetPosition

66 INDEX

Camera, 19	MeasureInMillis
GetRayDirections	Timer, 34
Camera, 19	metallic
GetSurfaceArea	Material, 25
AABB, 6	min
GetTarget	AABB, 7
Camera, 19	models
GetUp	Scene, 33
Camera, 19	normal
GetUsedThreadCount	HitPayload, 21
Renderer, 29	riitrayioau, 21
GetVerticalFovInDegrees	object
Camera, 20	BVHNode, 17
Hit	OnRayMiss
AccelerationStructure, 8	Renderer, 29
BVHNode, 14	OnResize
HitPayload, 21	Renderer, 29
material, 21	OnViewportResize
normal, 21	Camera, 20
t, 21	origin
ι, Ζ1	Ray, 26
Image, 22	, ==
~Image, 22	Position
GetDescriptor, 23	Camera, 20
Image, 22	
UpdateData, 23	Ray, 26
index	direction, 26
Material, 25	origin, <mark>26</mark>
IntersectsRay	RayDepth
AABB, 6	Renderer, 29
IsTerminating	Render
BVHNode, 14	Renderer, 30
	Renderer, 26
left	$\sim$ Renderer, 27
BVHNode, 17	Accelerate, 28
Light, 23	Accumulate, 28
GetObject, 24	Gamma, 28
Light, 23	GetAvailableThreadCount, 28
Sample, 24	GetFrameIndex, 28
	GetImage, 29
MakeHierarchyNaive	GetUsedThreadCount, 29
BVHNode, 15	OnRayMiss, 29
MakeHierarchySAH	OnResize, 29
BVHNode, 15	RayDepth, 29
Material, 24	Render, 30
albedo, 25	Renderer, 27
emissionPower, 25	Savelmage, 31
GetEmission, 25	SetUsedThreadCount, 31
index, 25	typ_t, 27
metallic, 25	UsedThreadCount, 32
roughness, 25	right
specular, 25	BVHNode, 17
material	roughness
HitPayload, 21	Material, 25
materials	Run Application 10
Scene, 33	Application, 10
max	Sample
AABB, 7	Campie

INDEX 67

```
BSDF, 11
    Light, 24
SaveImage
    Renderer, 31
Scene, 32
    boxes, 33
    camera, 33
    Deserialize, 33
    materials, 33
    models, 33
    Serialize, 33
    spheres, 34
    triangles, 34
Serialize
    Scene, 33
SetUsedThreadCount\\
    Renderer, 31
specular
    Material, 25
spheres
    Scene, 34
t
    HitPayload, 21
Target
    Camera, 20
Timer, 34
    MeasureInMillis, 34
triangles
    Scene, 34
typ_t
    Renderer, 27
Up
    Camera, 20
Update
    AccelerationStructure, 8
UpdateData
    Image, 23
UsedThreadCount
    Renderer, 32
VerticalFovInDegrees
    Camera, 21
```