```
Activating project at `~/Documents/Projects/AccelerateRT`
```

```
ingredients (generic function with 1 method)
```

```
ACC = Main.AccelerateRT.jl.AccelerateRT
```

```
BVH = Main.AccelerateRT.jl.AccelerateRT.BVH
```

```julia
using LinearAlgebra
```

```julia
using ProgressLogging
```

```julia
using Plots
```

```julia
using Logging
```

```julia
using Random
```

```
"Number of threads: 4"
```

Structures and Constants

```julia
mutable struct Camera
    pos::ACC.Vector3{Float32}        # position
    center::ACC.Vector3{Float32}     # center to look at
    fov::Float32                     # field of view in degrees
    res::ACC.Vector2{UInt32}         # rendered frame resolution
end
```

```julia
mutable struct Ray
    origin::ACC.Vector3{Float32}     # original position
    dir::ACC.Vector3{Float32}        # direction
    dist::Float32                    # distance to nearest hit
end
```

```
models = ["teapot", "bunny", "dragon", "sponza"]
```

```
bvhTypes = ["middle", "median", "sah"]
```

Utility Functions

```
intersect (generic function with 1 method)
```

`intersect!` (generic function with 1 method)

`loadData` (generic function with 1 method)

`rayTrace` (generic function with 1 method)

`visualize` (generic function with 1 method)

`visualizeAll` (generic function with 1 method)

`sample1` implements the following algorithm:

$$
\begin{aligned}
\theta &= \text{Uniform}(0, 2\pi), \\
\phi &= \text{Uniform}(0, \pi), \\
x &= \sin\phi \cdot \cos\theta, \\
y &= \sin\phi \cdot \sin\theta, \\
z &= \cos\phi.
\end{aligned}
$$

`sample1` (generic function with 3 methods)

`sample2` implements the following algorithm by Marsaglia (1972):

$$
\begin{aligned}
x_1, x_2 &= \text{Uniform}(-1, 1), \\
\text{Reject If} \quad & x_1^2 + x_2^2 \geq 1, \\
x &= 2x_1\sqrt{1 - x_1^2 - x_2^2}, \\
y &= 2x_2\sqrt{1 - x_1^2 - x_2^2}, \\
z &= 1 - 2(x_1^2 + x_2^2).
\end{aligned}
$$

`sample2` (generic function with 3 methods)

`sample3` implements algorithm by Cook (1957):

$$
\begin{aligned}
x_0, x_1, x_2, x_3 &= \text{Uniform}(-1, 1), \\
\text{Reject If} \quad & x_0^2 + x_1^2 + x_2^2 + x_3^2 \geq 1, \\
x &= \frac{2(x_1x_3 + x_0x_2)}{x_0^2 + x_1^2 + x_2^2 + x_3^2}, \\
y &= \frac{2(x_2x_3 - x_0x_1)}{x_0^2 + x_1^2 + x_2^2 + x_3^2}, \\
z &= \frac{x_0^2 + x_3^2 - x_1^2 - x_2^2}{x_0^2 + x_1^2 + x_2^2 + x_3^2}.
\end{aligned}
$$

```
sample3 (generic function with 3 methods)
```

`sample4` implements a simple algorithm with Gaussian distribution:

$$
\begin{gathered}
x, y, z = \text{Normal}(), \\
\vec{v} = \frac{1}{\sqrt{x^2 + y^2 + z^2}} \begin{bmatrix} x \\ y \\ z \end{bmatrix}
\end{gathered}
$$

```
sample4 (generic function with 3 methods)
```

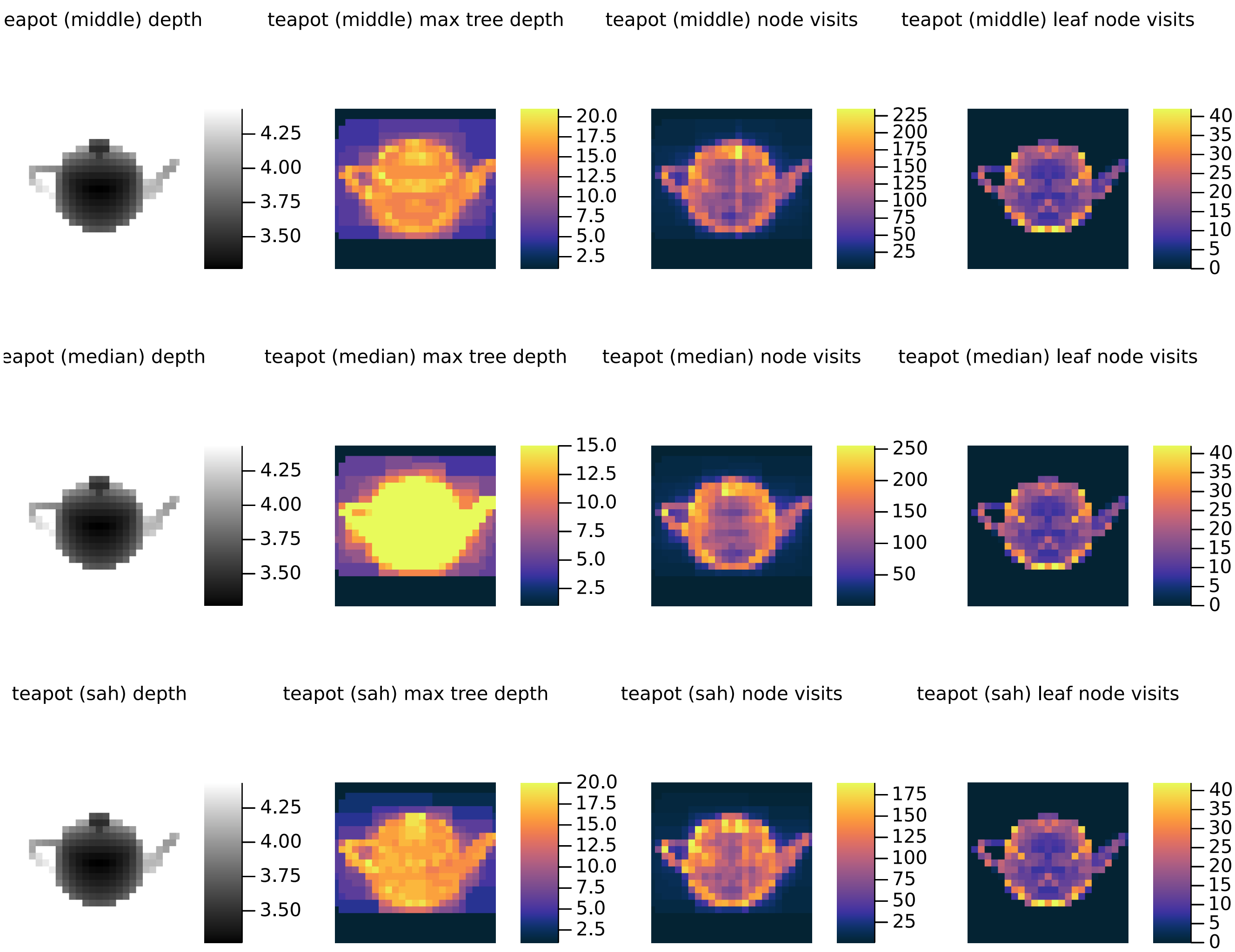
```
visualizeSphere (generic function with 1 method)
```

Comparison

Compare different BVH's on teapot model

```
camera =
  Camera(StaticArrays.MVector{3, Float32}: [0.0, 3.0, 3.0], StaticArrays.MVector{3, Float32
```

```
camera = Camera(
    ACC.Vector3{Float32}(0, 3, 3),
    ACC.Vector3{Float32}(0, 0, 0),
    45.0f0,
    ACC.Vector2{UInt32}(25, 25)
)
```




```
visualizeAll(camera, models[1])
```

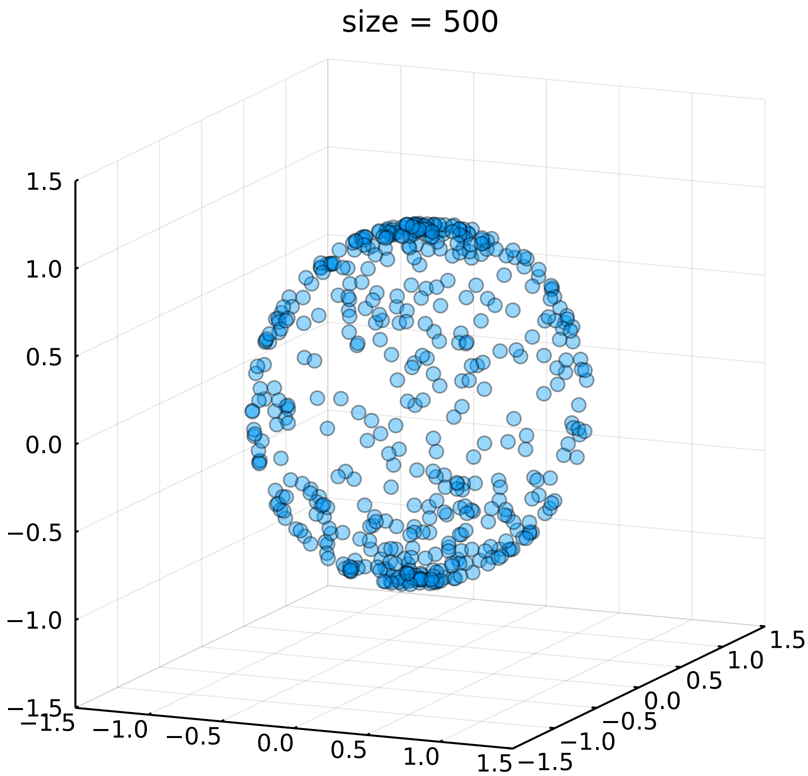
1 1 100%

Sampling

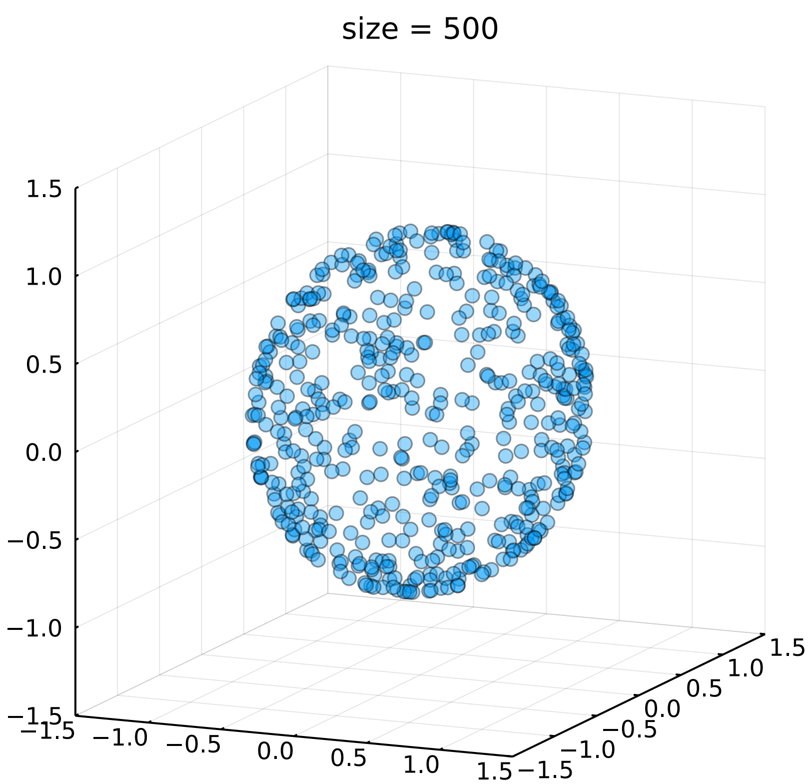
Test unit sphere sampling for experiment

`https://mathworld.wolfram.com/SpherePointPicking.html` for reference

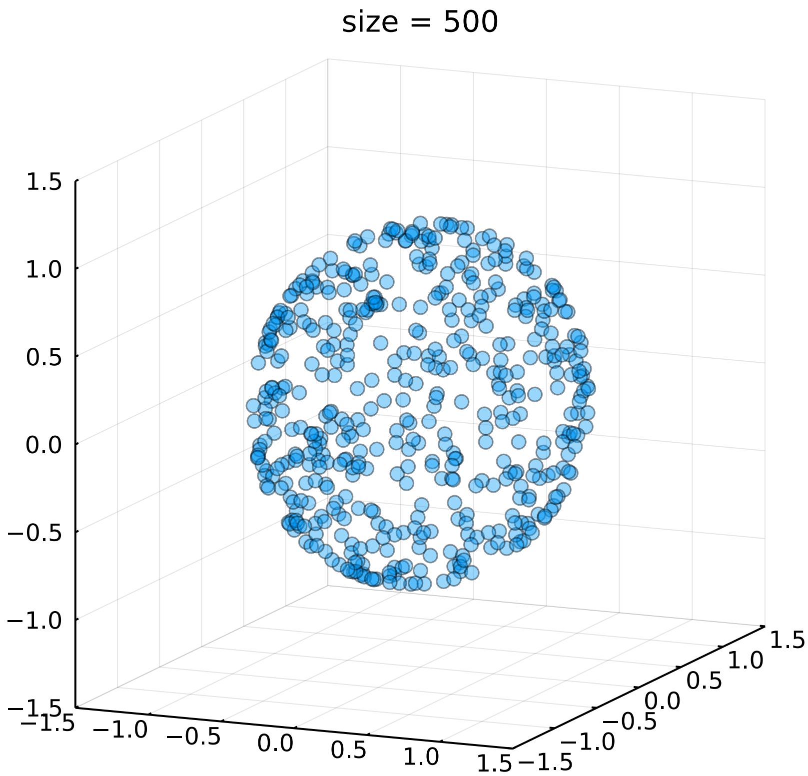
```
randseed = 123
```



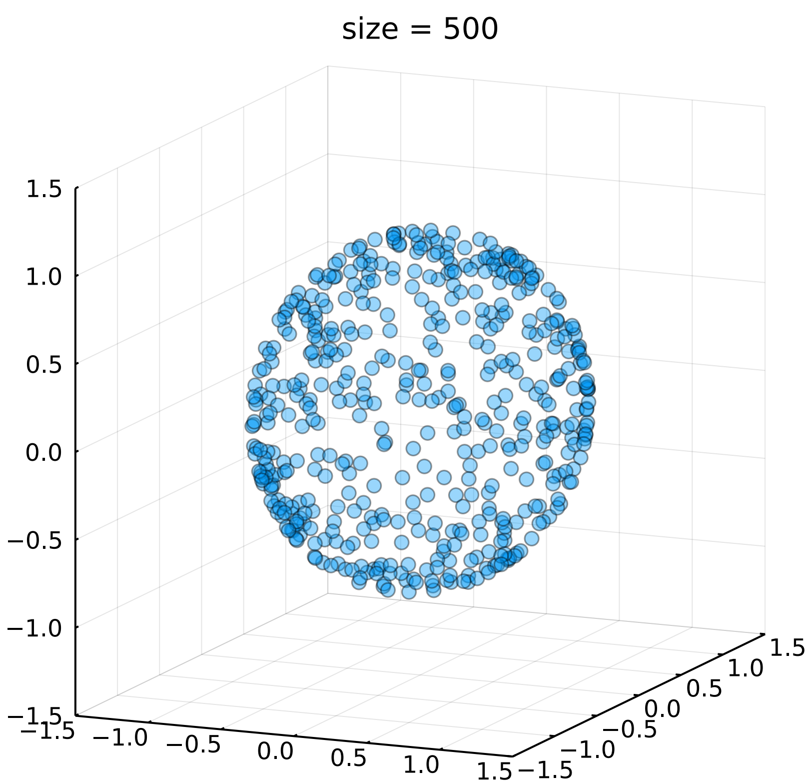

```
visualizeSphere(sample1(500, randseed))
```




```
visualizeSphere(sample2(500, randseed))
```




```
visualizeSphere(sample3(500, randseed))
```




```
visualizeSphere(sample4(500, randseed))
```