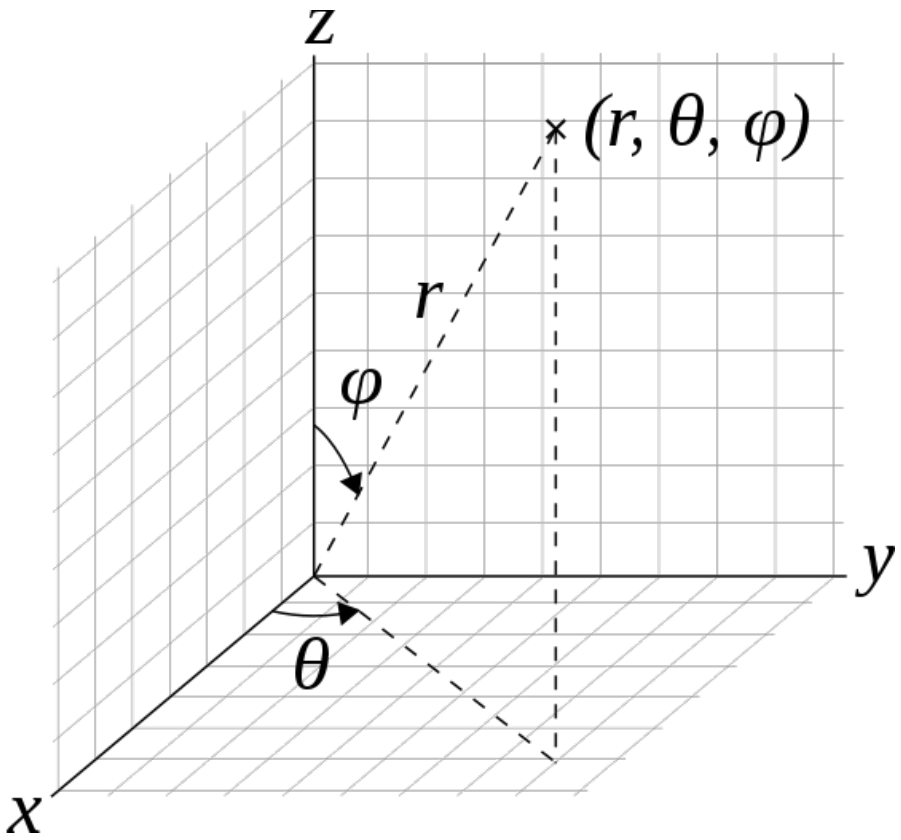
uniformly distributed sampling on a sphere

Chuan YU

We want to generate uniformly distributed points on a sphere.



Given polar angle $\phi \in (0, \pi)$ and azimuthal angle $\theta \in (0, 2\pi)$, we have:

$$x = rsin(\phi)cos(\theta)$$
$$y = rsin(\phi)sin(\theta)$$
$$z = rcos(\phi)$$

Let v be a point on the unit sphere S. We want probability density $p(v)$ to be a constant. Since $\int_S p(v)dA = 1$, we obtain:

$$p(v) = \frac{1}{4\pi}$$

We want to represent point v with spherical coordinates θ and ϕ and find the probability density function $p(\phi, \theta)$:

$$p(v)dA = \frac{1}{4\pi}dA = p(\phi, \theta)d\phi d\theta$$

Since $dA = sin\phi d\phi d\theta$, we have:

$$p(\phi, \theta) = \frac{sin\phi}{4\pi}$$

We can get probability density function of ϕ and θ respectively:

$$p(\phi) = \int_0^{2\pi} p(\phi, \theta) d\theta = \int_0^{2\pi} \frac{sin\phi}{4\pi} d\theta = \frac{sin\phi}{2}$$

$$p(\theta) = \int_0^{\pi} p(\phi, \theta) d\phi = \int_0^{\pi} \frac{sin\phi}{4\pi} d\phi = \frac{1}{2\pi}$$

We find $p(\phi)$ scales with $sin\phi$ and θ is a uniformly distributed variable.

The algorithm for sampling the distribution $p(\phi)$ using inverse transform sampling is as follows:

- Generate a uniform random number ζ from the distribution $U[0, 1]$
- Compute ϕ such that cumulative distribution function $F(\phi) = \zeta$, i.e. $F^{-1}(\zeta)$
- This ϕ is a random number from the distribution $f(\phi)$

Based on inverse transform sampling, we need cumulative distribution function of ϕ

$$F(\phi) = \int_0^{\phi} p(\phi') d\phi' = \int_0^{\phi} \frac{sin\phi'}{2} d\phi' = \frac{1}{2}(1 - cos\phi)$$

We can obtain ϕ:

$$\phi = arccos(1 - 2\zeta)$$

The algorithm below in C++ shows how to generate uniformly distribution points on a sphere using this method:

```
#include <chrono>
#include <random>
#include <math.h>
#include <iostream>

int main(int argc, char *argv[]){
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::mt19937 generator(seed);
    std::uniform_real_distribution<double> uniform01(0.0, 1.0);

    int n = 10;

    for (int i = 0; i < n; ++i){
        double phi = acos(1 - 2 * uniform01(generator));
        double theta = 2 * M_PI * uniform01(generator);
        double x = sin(phi) * cos(theta);
        double y = sin(phi) * sin(theta);
        double z = cos(phi);

        std::cout << theta << ", " << phi << ", " << x << ", " << y << ", " << z <<
            std::endl;
    }

    return 0;
}
```

According to the algorithm above, the probability density function of θ is constant, we only need to verify whether points are uniform along ϕ axis in spherical coordiantes. We first choose the region with z-coordiate $>= z$, and count the number of points in it. Then we rotate this region $\frac{\pi}{m}$ angle along X-axis and count the number of points in it. Repeat to rotate region and count the number. Finally we compare the numbers and plot the figure.

Before we do the verification process, we need to figure out how to count the number of points in the rotated regions. Instead of rotating the region, we rotate the coordiantes using the following formula:

$$\begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix}$$

After getting the rotated coordiantes, we can simply count the points with z_{new}-coordiate $>= z$.

Therefore our verification process is as follows:

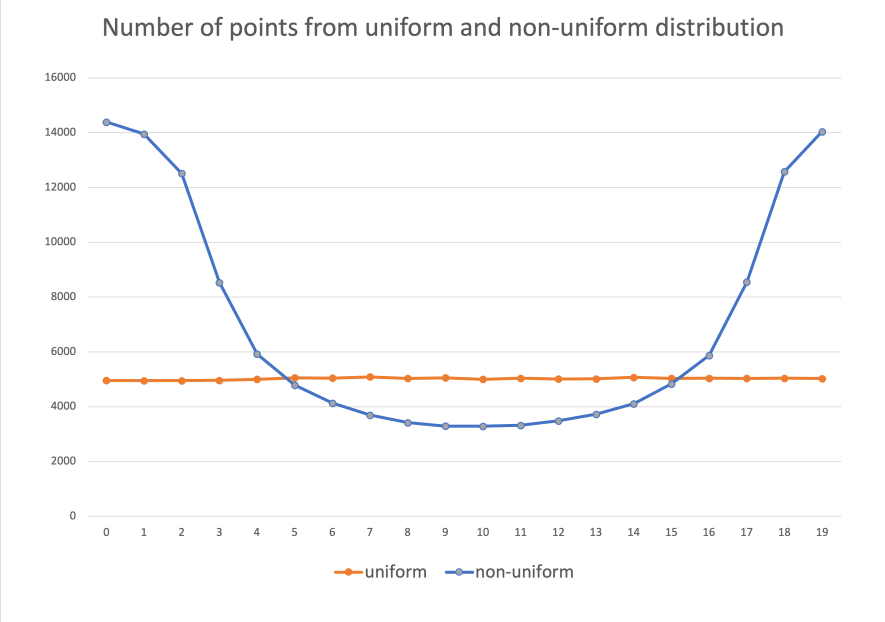
1. Generate n points based on algorithm above
2. Count the number of points with z-coordiate $>= z$
3. Rotate the coordiate $\frac{\pi}{m} \times i$ angle along X-axis $(i = 0 \dots m-1)$
4. Compute the new coordiates of points after rotation
5. Repeat Step 2 to Step 4 m times
6. Compare m numbers

In order to compare our algorithm with non-uniform algorithm, we use another algorithm using:

$$\phi = \pi \times \zeta$$

ζ is a uniform random number from the distribution $U[0, 1]$

We use $n = 100000$, $m = 20$, $z = 0.9$ and plot m numbers with i



We find that points generated from our algorithm are relatively uniform.

C++ code including verification is as follows.
Note: in Line 26 and 35, we can use reference to vector instead of vector itself to eliminate copy constructor, but there are bugs that latex cannot recognize reference symbol:

```
1  #include <chrono>
2  #include <random>
3  #include <math.h>
4  #include <iostream>
5
6  std::vector<std::vector<double>> uniform_points(int num){
7      double phi, theta;
8      std::vector<std::vector<double>> points(num);
9
10     unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
11     std::mt19937 generator(seed);
12     std::uniform_real_distribution<double> uniform01(0.0, 1.0);
13
14     for (int i = 0; i < num; ++i){
15         phi = acos(1 - 2 * uniform01(generator));
16         theta = 2 * M_PI * uniform01(generator);
17
18         points[i].push_back(sin(phi) * cos(theta));
19         points[i].push_back(sin(phi) * sin(theta));
20         points[i].push_back(cos(phi));
21     }
22
23     return points;
24 }
25
26 int count_points(std::vector<std::vector<double>> points, double z){
27     int count = 0;
28     for (auto point: points){
29         if (point[2] >= z)
30             ++count;
31     }
32     return count;
33 }
34
35 std::vector<std::vector<double>> rotate_x(std::vector<std::vector<double>> points,
       double theta){
36     int size = points.size();
37     std::vector<std::vector<double>> rotate_points(size);
38
39     double x, y, z;
40
41     for (int i = 0; i < size; ++i){
42         x = points[i][0];
43         y = points[i][1];
44         z = points[i][2];
45
46         rotate_points[i].push_back(x);
47         rotate_points[i].push_back(y * cos(theta) - z * sin(theta));
48         rotate_points[i].push_back(y * sin(theta) + z * cos(theta));
49     }
50
51     return rotate_points;
52 }
53
54 std::vector<std::vector<double>> not_uniform_points(int num){
```

```
55      double phi, theta;
56      std::vector<std::vector<double>> points(num);
57
58      unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
59      std::mt19937 generator(seed);
60      std::uniform_real_distribution<double> uniform01(0.0, 1.0);
61
62      for (int i = 0; i < num; ++i){
63          phi = M_PI * uniform01(generator);
64          theta = 2 * M_PI * uniform01(generator);
65
66          points[i].push_back(sin(phi) * cos(theta));
67          points[i].push_back(sin(phi) * sin(theta));
68          points[i].push_back(cos(phi));
69      }
70
71      return points;
72  }
73
74  int main(int argc, char *argv[]){
75      int n = 100000;
76      int m = 20;
77      double z = 0.9;
78
79      std::vector<std::vector<double>> points;
80      std::vector<std::vector<double>>rotate_points;
81
82      std::cout << "uniform:" << std::endl;
83      points = uniform_points(n);
84      for (int i = 0; i <= m - 1; ++i){
85          rotate_points = rotate_x(points, M_PI / m * i);
86          std::cout << count_points(rotate_points, z) << std::endl;
87      }
88
89      std::cout << "non-uniform:" << std::endl;
90      points = not_uniform_points(n);
91      for (int i = 0; i <= m - 1; ++i){
92          rotate_points = rotate_x(points, M_PI / m * i);
93          std::cout << count_points(rotate_points, z) << std::endl;
94      }
95
96      return 0;
97  }
```