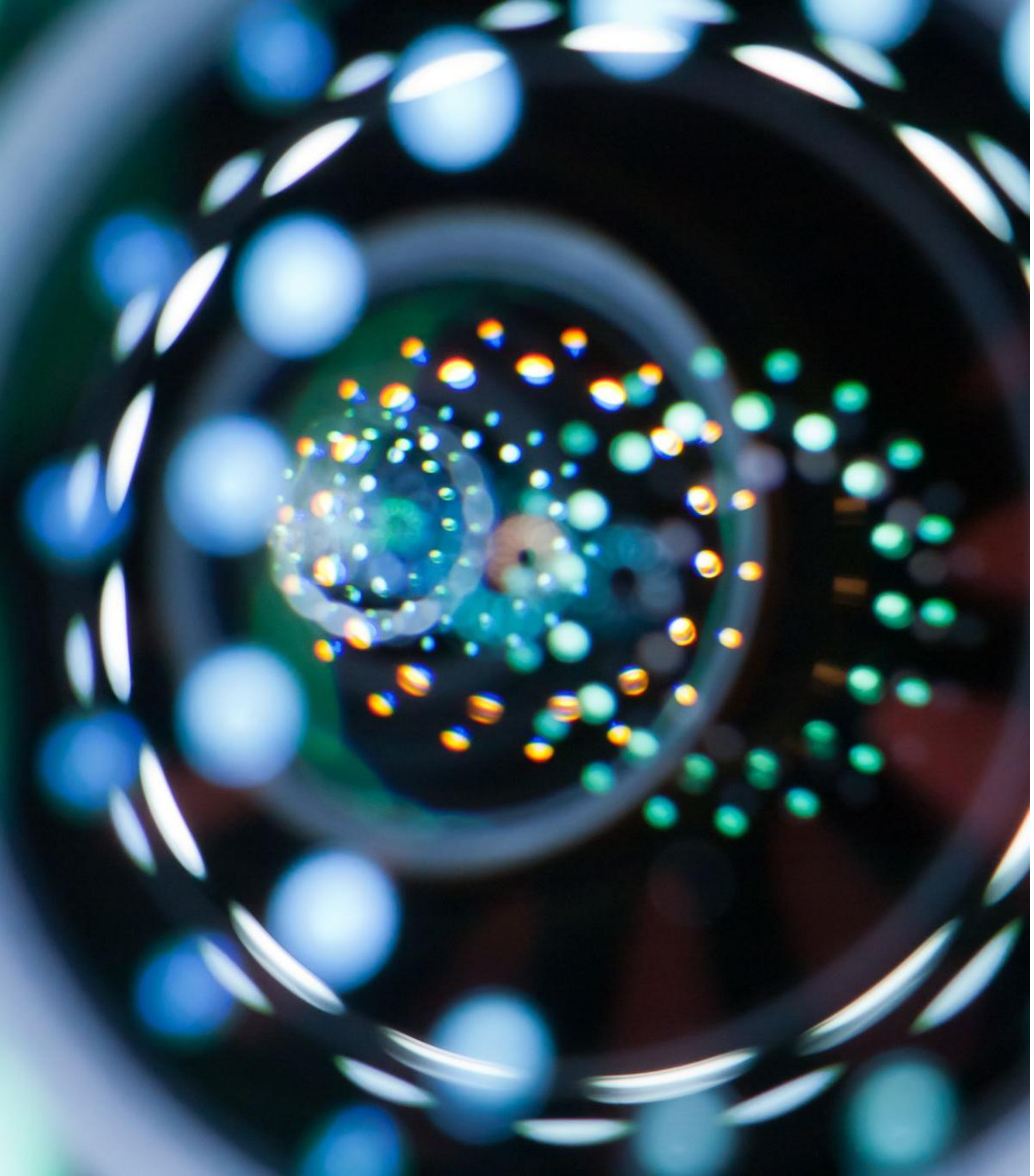


Ray Tracing in Entertainment Industry

Tanaboon Tongbuasirilai
Dept. Computer Science
Kasetsart University

Week 9

Sampling techniques and reconstruction

Sampling techniques and reconstruction

What sampling techniques do we have so far ?

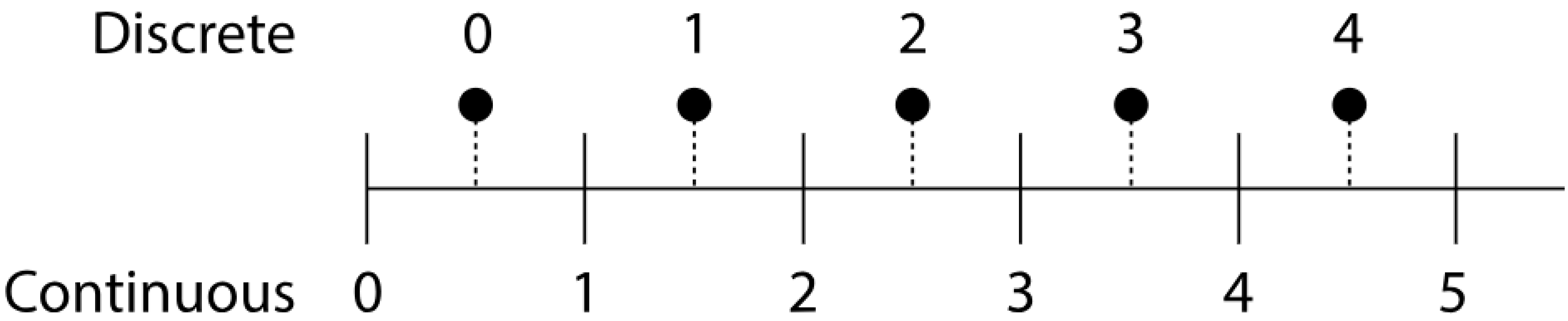
- Random sampling of a 3D vector : random_vec3()
- Random unit vector in a unit disk : random_vec3_in_unit_disk()
- Random unit vector in a unit sphere : random_vec3_unit ()
- Random directions in scattering()
- Random locations in a pixel

How random samples impact to the ray tracer ?

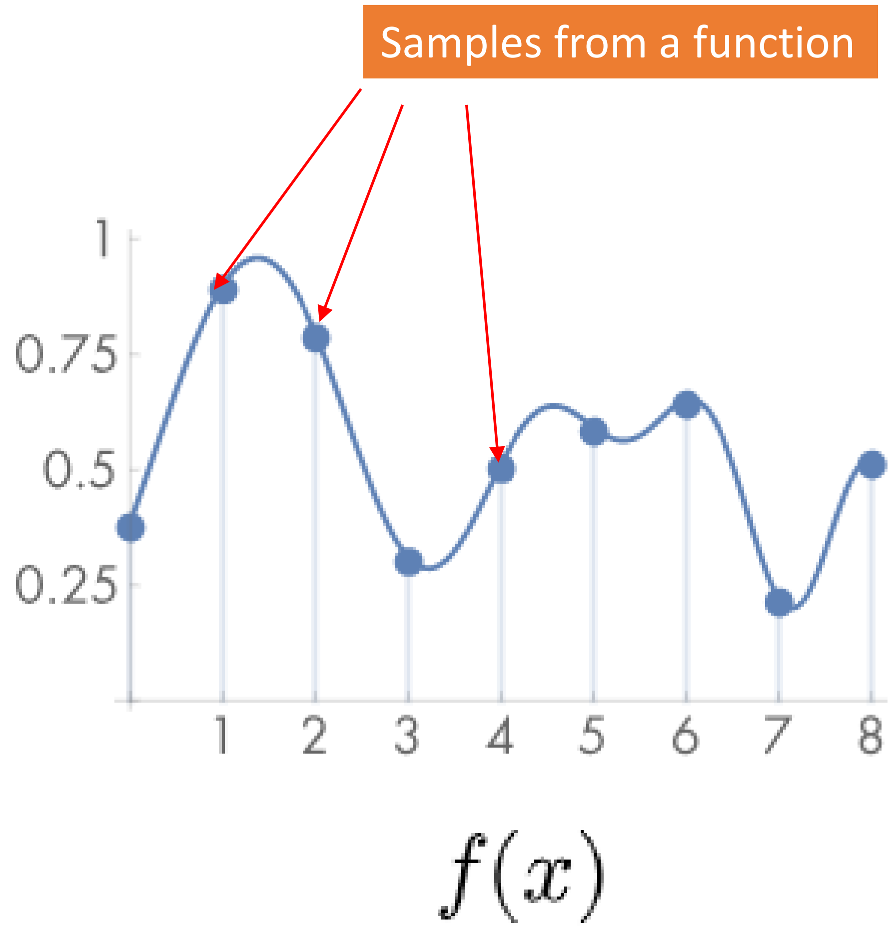
- Camera
- Material
- Light
- Integrator

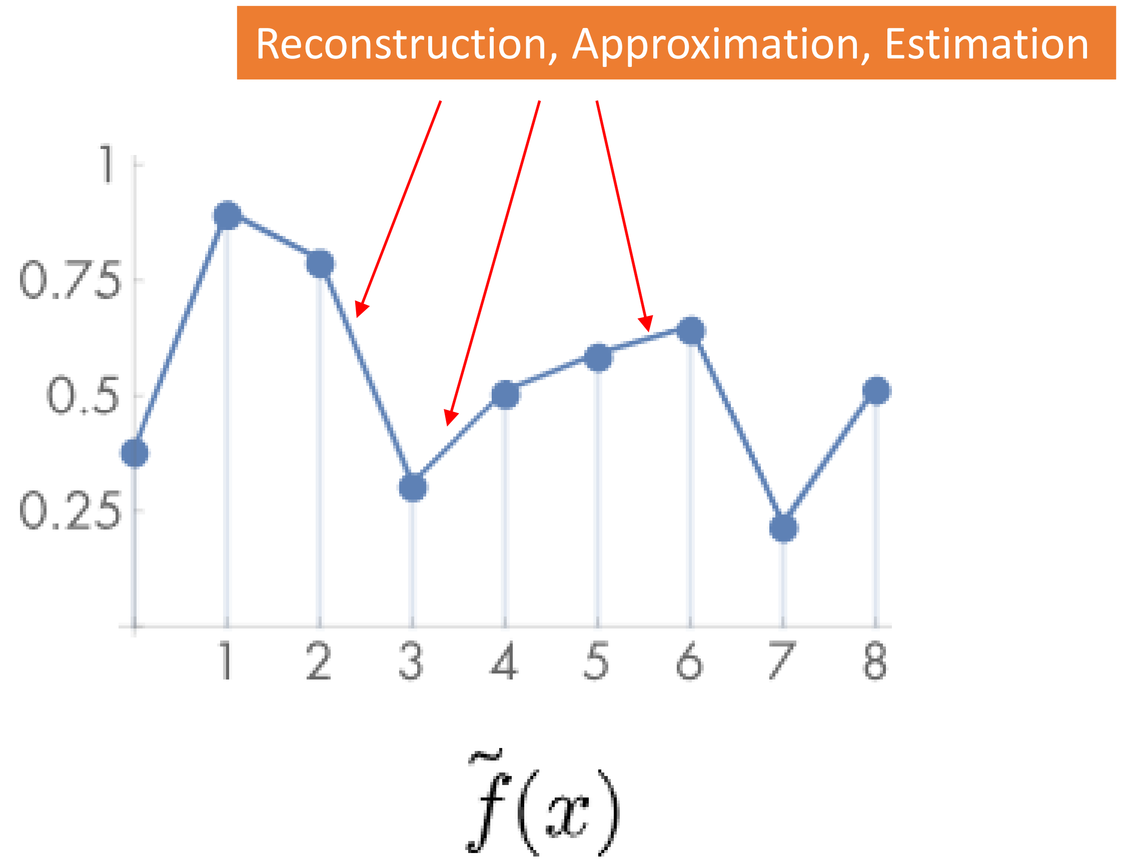
Why do we need sampling ?

- Discrete vs. Continuous
  - Image with different resolutions
- Functions is defined on Real (continuous) domain.
- We would like to represent continuous functions in an efficient way.

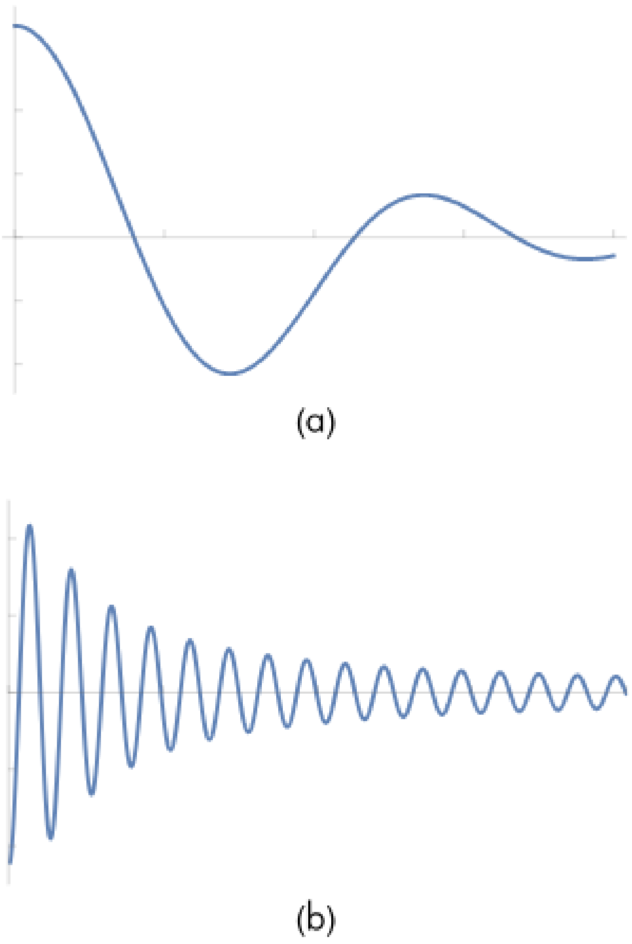


Sampling and reconstruction

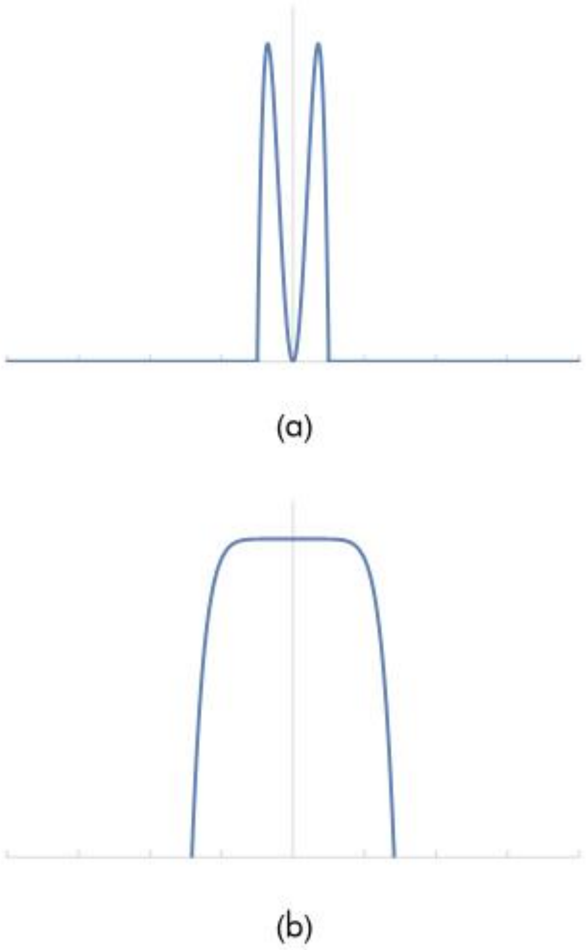




High-frequency vs. Low-frequency

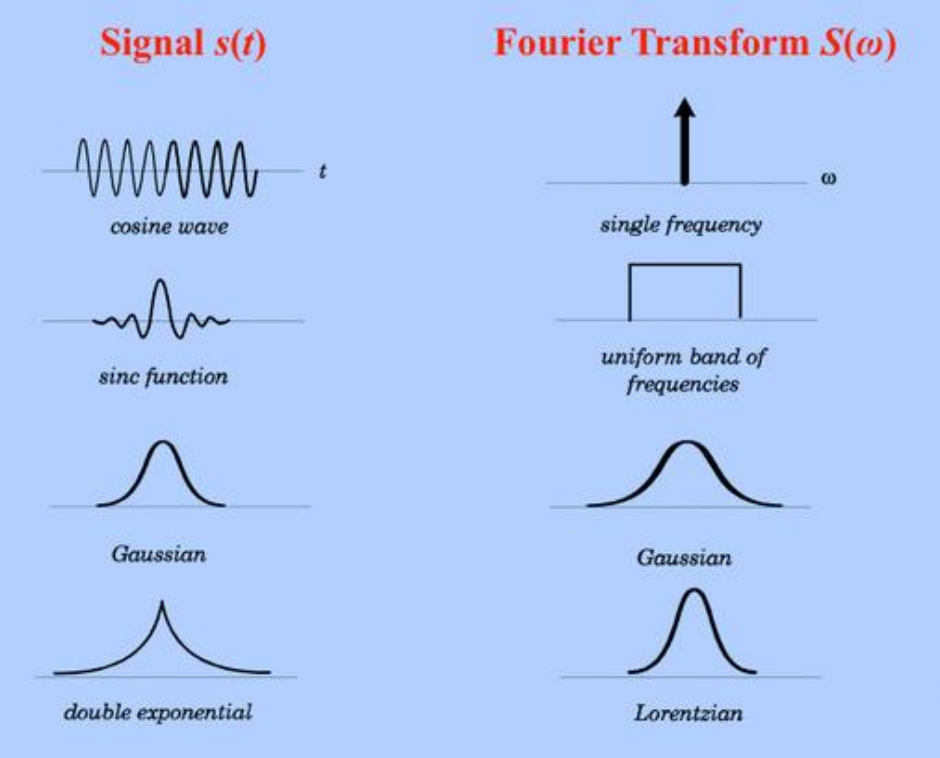


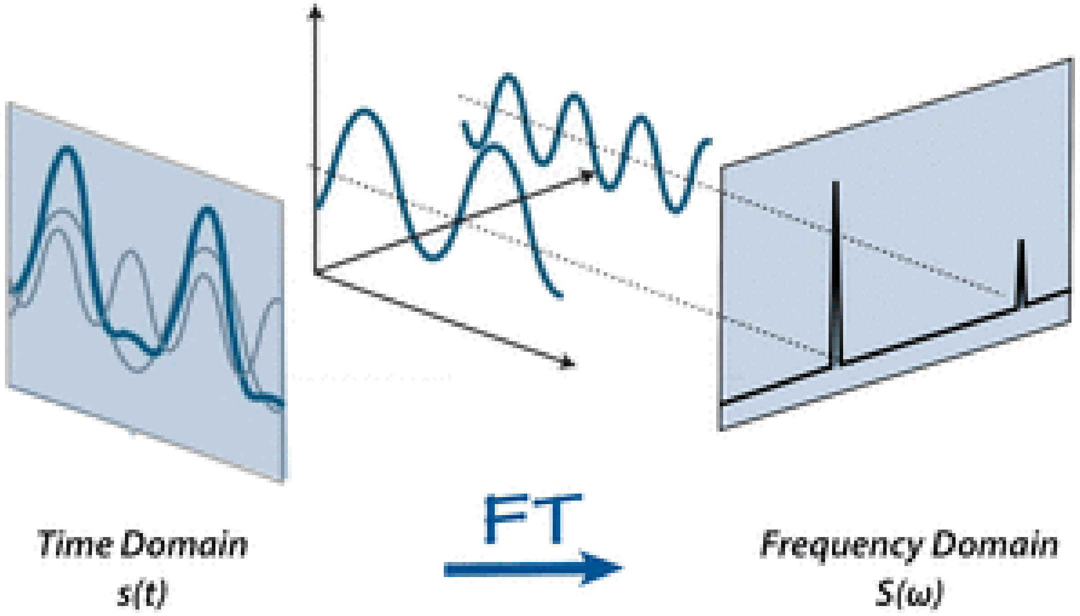
Spatial domain



Frequency domain

Fourier transform





mriquestions.com/fourier-transform-ft.html

Fourier transform of a 1D function

$$F(\omega) = \int_{-\infty}^{\infty} f(x)\, \mathrm{e}^{-\mathrm{i}2\pi\omega x}\, \mathrm{d}x.$$

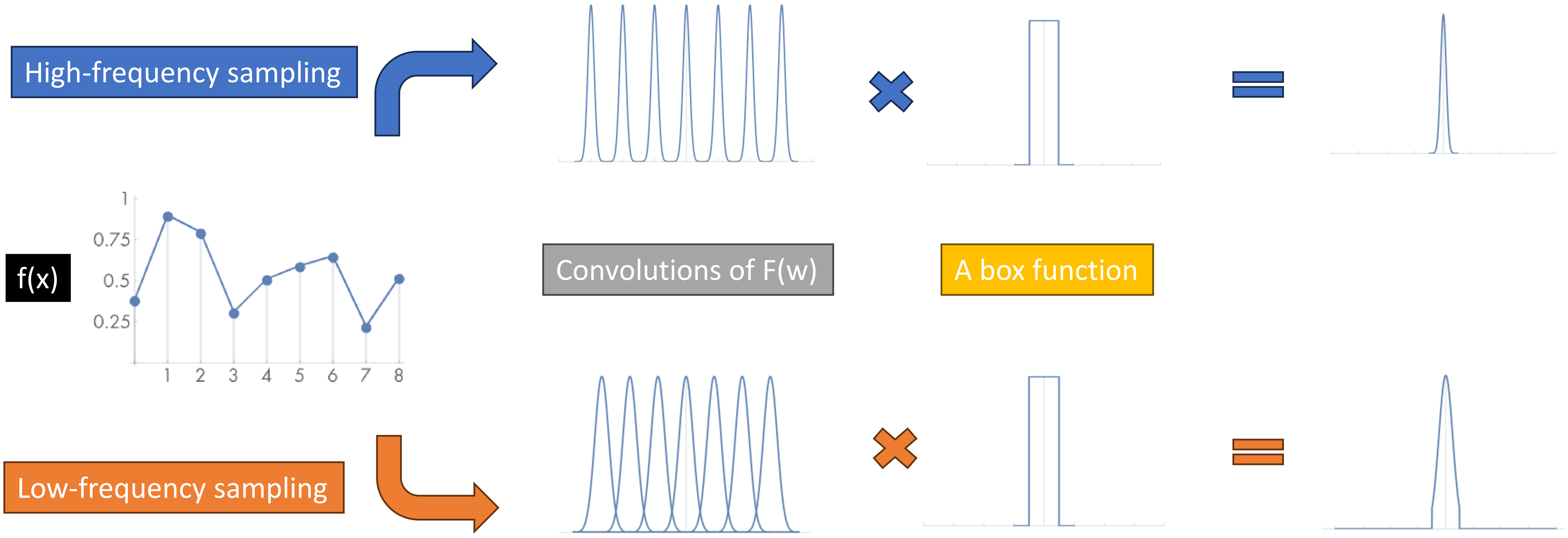
Fourier transform

$$f(x) = \int_{-\infty}^{\infty} F(\omega)\, \mathrm{e}^{\mathrm{i}2\pi\omega x}\, \mathrm{d}\omega.$$

Inverse Fourier transform

Aliasing

F(w) = Fourier transform of f(x)



What it means to image synthesis.

We would like to reconstruct the function representing the 3D scene.
Aliasing occurs naturally for such problem and there are many parameters involving in this issue.

What to write in the image plane.

What we consider in sampling.
- Image resolution
- Time
- Lens
- etc.

Radiance contribution at the location (x,y).

$$f(x, y) \to L.$$

Antialiasing techniques

Nonuniform sampling

Nonuniform sampling tends to turn the regular aliasing artifacts into noise, which is less distracting to the human visual system.

Adaptive sampling

If we can identify the regions where they need more samples, it will be less expensive than taking more samples in every region.

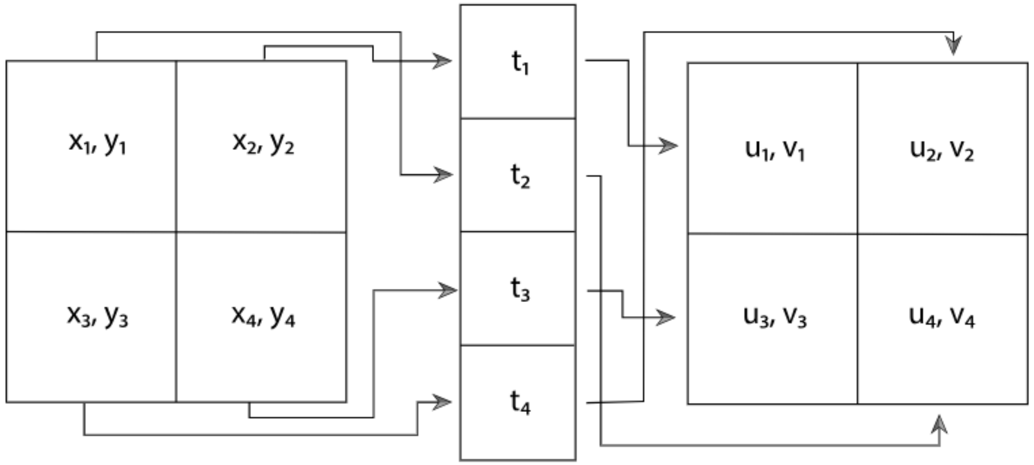
Prefiltering

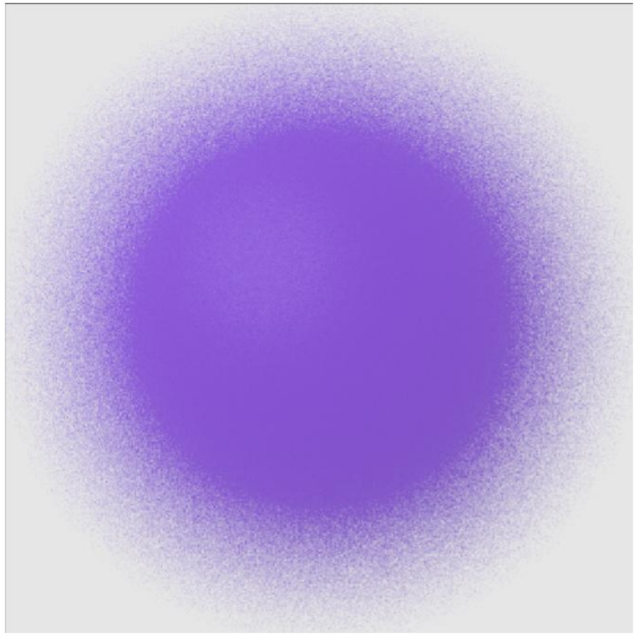
Filtering can be used to mitigate aliasing by lower frequencies so that the current sampling rate cannot capture the aliasing.

Sampling over image plane

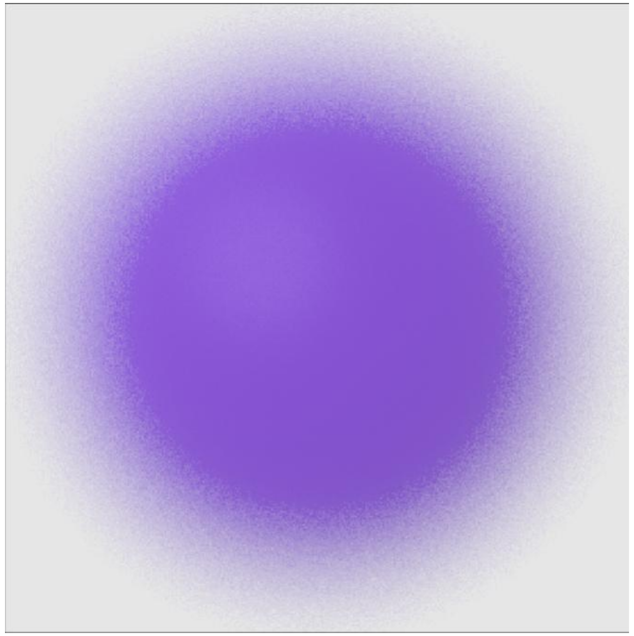
Stratified sampling

- Subdividing region into smaller sub regions.
- Each sub regions is called strata.
- The more sub regions there is, the higher the sampling rate is.
- Each sample variable is drawn independently to each other in high-dimensional space.
    - This turns the aliasing problem into a noisy pattern.
    - The drawn samples can cover the sample space without excessive computation.





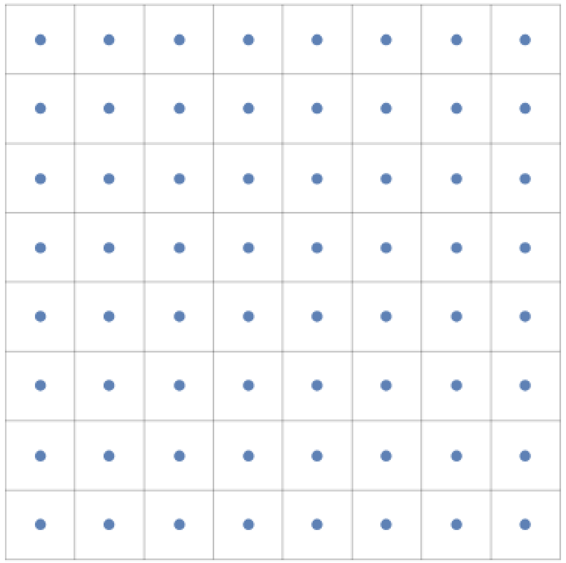
Random



Stratified

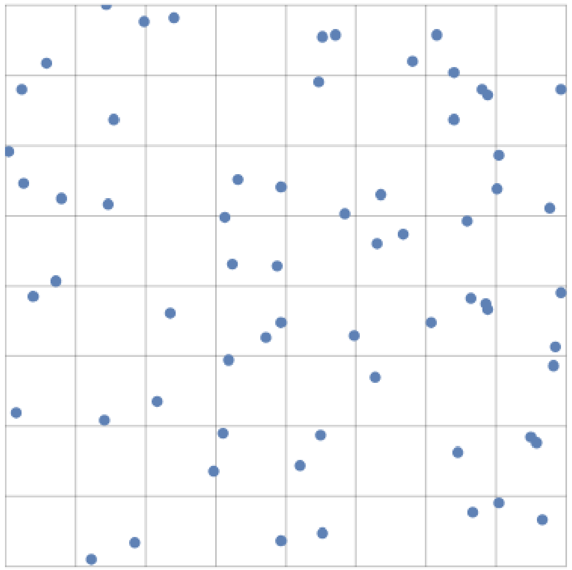
Comparison

Comparison



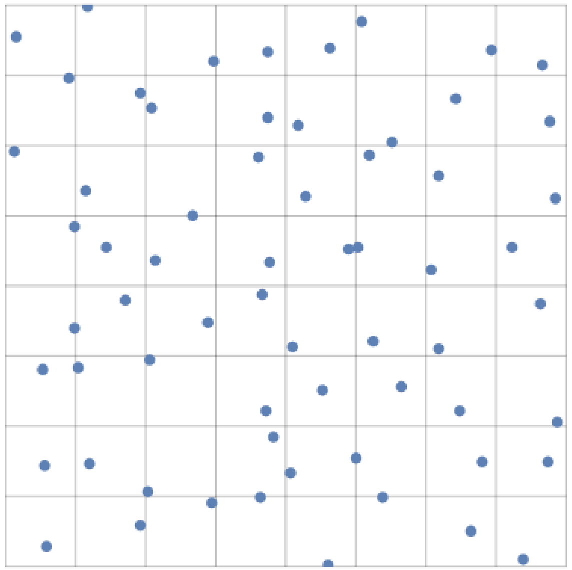
Uniform stratified pattern

subdividing regions but still uniform sampling



Uniform random pattern

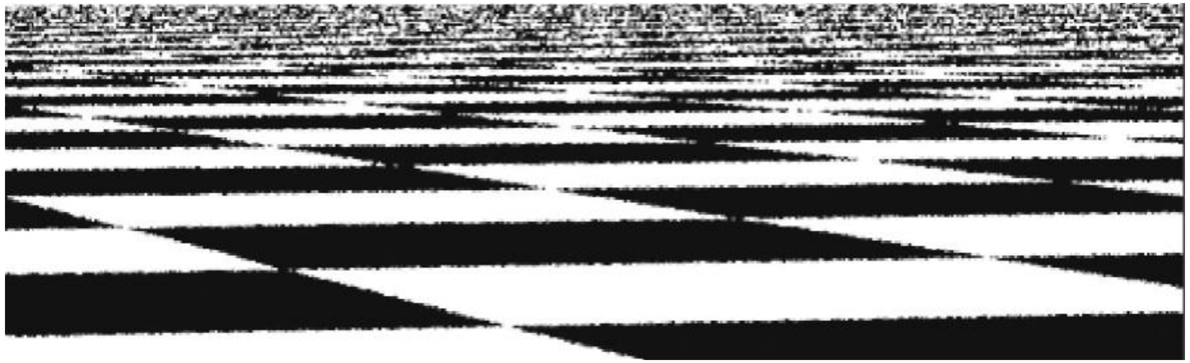
uniform random the whole region



A stratified jittered pattern

subdividing regions and jiterring each subregion

Jittered vs. Unjittered



Coding Renderer

```python
def render(self):
    # gather lights to the light list
    self.scene.find_lights()

    for j in range(self.camera.img_height):
        for i in range(self.camera.img_width):

            pixel_color = rtu.Color(0,0,0)
            # shoot multiple rays at random locations inside the pixel
            for spp in range(self.camera.samples_per_pixel):
                generated_ray = self.camera.get_ray(i, j)
                pixel_color = pixel_color + self.integrator.compute_scattering(generated_ray, self.scene, self.camera.max_depth)

            self.camera.write_to_film(i, j, pixel_color)
```

```python
def render_jittered(self):
    # gather lights to the light list
    self.scene.find_lights()
    sqrt_spp = int(math.sqrt(self.camera.samples_per_pixel))

    for j in range(self.camera.img_height):
        for i in range(self.camera.img_width):

            pixel_color = rtu.Color(0,0,0)
            # shoot multiple rays at random locations inside the pixel
            for s_j in range(sqrt_spp):
                for s_i in range(sqrt_spp):

                    generated_ray = self.camera.get_jittered_ray(i, j, s_i, s_j)
                    pixel_color = pixel_color + self.integrator.compute_scattering(generated_ray, self.scene, self.camera.max_depth)

            self.camera.write_to_film(i, j, pixel_color)
```

Coding Camera

```
def get_jittered_ray(self, i, j, s_i, s_j):

    pixel_center = self.pixel00_location + (self.pixel_du*i) + (self.pixel_dv*j)
    pixel_sample = pixel_center + self.pixel_sample_square(s_i, s_j)

    ray_origin = self.center
    ray_direction = pixel_sample - ray_origin

    return rtr.Ray(ray_origin, ray_direction)
```

```
def pixel_sample_square(self, s_i, s_j):
    px = -0.5 + self.one_over_sqrt_spp * (s_i + rtu.random_double())
    py = -0.5 + self.one_over_sqrt_spp * (s_j + rtu.random_double())
    return (self.pixel_du * px) + (self.pixel_dv * py)
```

Sampling over directions

Sampling a direction on hemisphere

It might take a long time to get the correct result.

```
@staticmethod
def random_vec3_on_hemisphere(vNormal):
    in_unit_sphere = Vec3.random_vec3_unit()
    if Vec3.dot_product(in_unit_sphere, vNormal) > 0.0:
        return in_unit_sphere
    else:
        return -in_unit_sphere
```

A Las Vegas algorithm is a randomized algorithm that always gives correct results.

A Monte Carlo algorithm is a randomized algorithm whose output may be incorrect, but it will improve when the samples are increased.

yourbasic.org/algorithms/las-vegas/

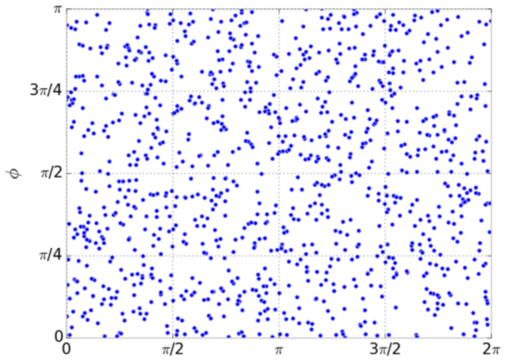
Scattered directions

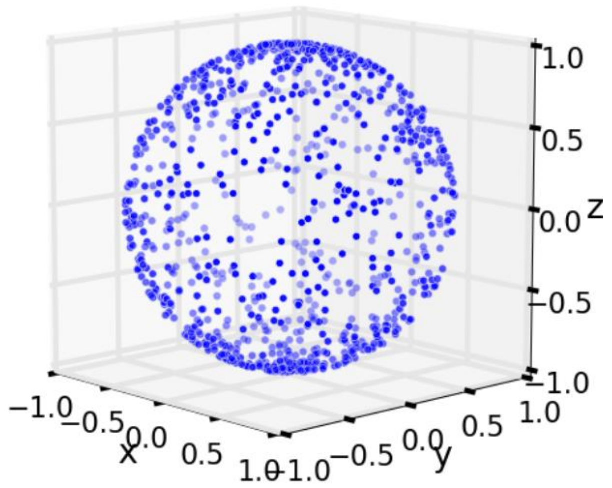
This may drop our sampling performance.

```python
def scattering(self, rRayIn, hHinfo):
    reflected_direction = -hHinfo.getNormal()
    # check if the reflected direction is below the surface normal
    while rtu.Vec3.dot_product(reflected_direction, hHinfo.getNormal()) <= 1e-8:

        # compute scattered ray
        reflected_direction = hHinfo.getNormal() + rtu.Vec3.random_vec3_unit()
        if reflected_direction.near_zero():
            reflected_direction = hHinfo.getNormal()

    reflected_ray = rtr.Ray(hHinfo.getP(), reflected_direction)
    phong_color = self.BRDF(rRayIn, reflected_ray, hHinfo)

    return rtu.Scatterinfo(reflected_ray, phong_color)
```

Sampling a direction in a unit sphere –
Generating random directions relative to the Z axis

$\theta \in (0,90)$
$\varphi \in (0,180)$



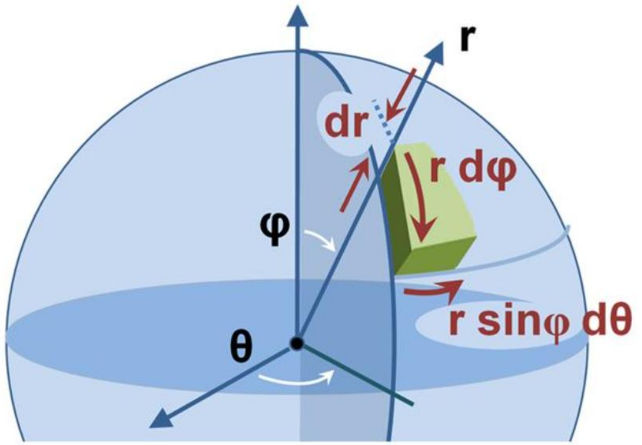


A direction from spherical coordinate to cartesian coordinates

$$x = \cos(\phi) \cdot \sin(\theta)$$

$$y = \sin(\phi) \cdot \sin(\theta)$$

$$z = \cos(\theta)$$



The issue is the differential surface.

corysimon.github.io/articles/uniformdistn-on-sphere/

A more effective sampling approach

Cosine Sampling

Uniform sampling on …

Sphere

$$x = \cos(2\pi r_1) \cdot 2\sqrt{r_2(1 - r_2)}$$

$$y = \sin(2\pi r_1) \cdot 2\sqrt{r_2(1 - r_2)}$$

$$z = 1 - 2r_2$$

$r1 \ \in [0,1]$
$r2 \ \in [0,1]$

hemisphere

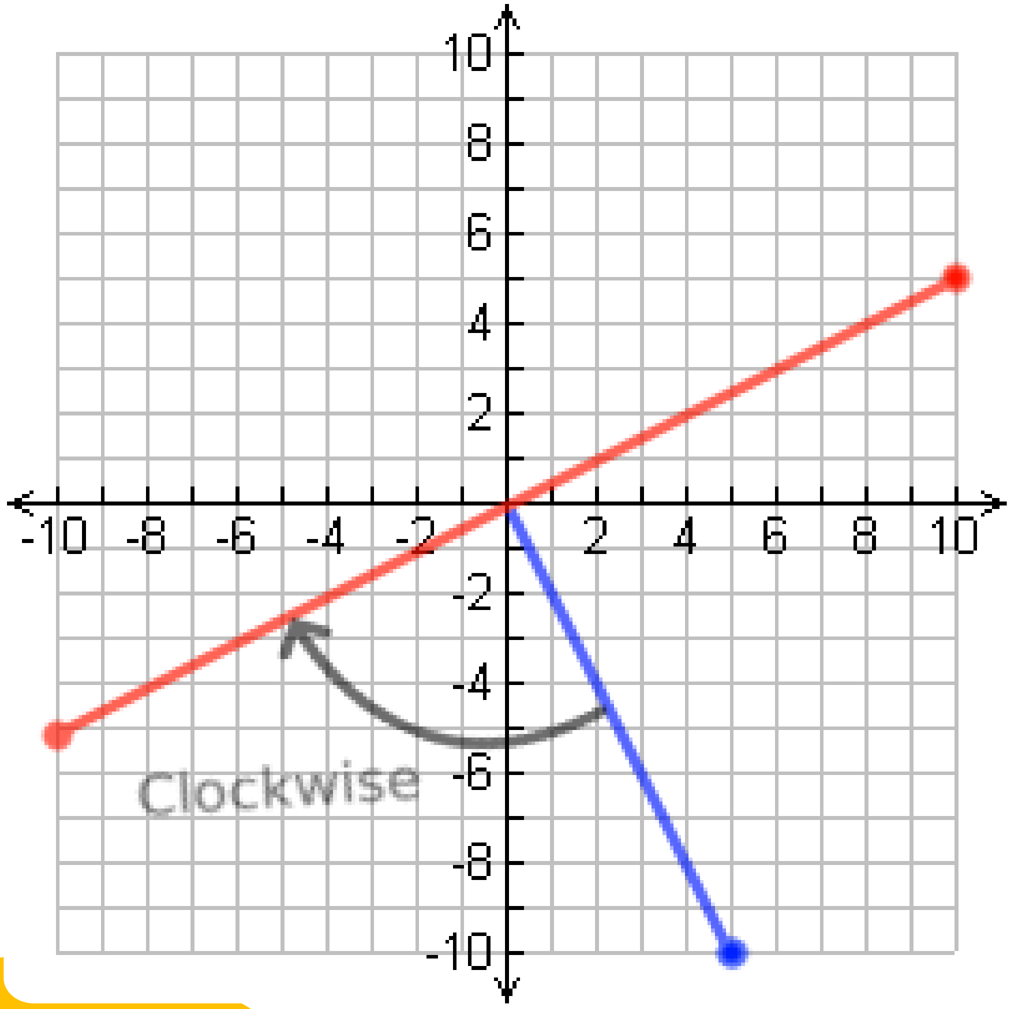
$$z = \cos(\theta) = \sqrt{1 - r_2}$$

$$x = \cos(\phi)\sin(\theta) = \cos(2\pi r_1)\sqrt{1 - z^2} = \cos(2\pi r_1)\sqrt{r_2}$$

$$y = \sin(\phi)\sin(\theta) = \sin(2\pi r_1)\sqrt{1 - z^2} = \sin(2\pi r_1)\sqrt{r_2}$$

$r1 \ \in [0,1]$
$r2 \ \in [0,1]$

Before we continue further on sampling, Let me introduce you a useful tool for handling a change of coordinate system.



Orthonormal bases

- Two important properties for this.
  - Orthogonality
    - The vectors are all pairwise perpendicular, meaning their dot product is zero.
  - Normality
    - Each vector has a magnitude (length) of 1.

Why ?

- Scene, objects, rays, cameras must be aligned in the same coordinate system.
- Cartesian coordinate system (x, y, z) is an example of a subset of orthonormal basis.
- Sometimes it is difficult to do operations on the original coordinate system, like Cartesian coordinate system.
- Generating an orthonormal basis is a way to handle arbitrary points on the original coordinate system.

Relative coordinates

$$\text{Location is } \mathbf{O} + \underbrace{3\mathbf{x} - 2\mathbf{y} + 7\mathbf{z}}_{\text{Relative point to the origin}}$$

Point of origin

Relative point to the origin

$$\boxed{\text{Location is } \mathbf{O}' + u\mathbf{u} + v\mathbf{v} + w\mathbf{w}}$$

A point with coefficients (*u,v,w*) on U,V,W coordinate system relative to the origin O'.

Generating an orthonormal basis given that we have a normal vector

Pick an arbitrary axis, ***a***, not parallel to the normal, ***n***.

Find a vector ***s*** perpendicular to ***a*** and ***n***.

Find a vector ***t*** perpendicular to ***s*** and ***n***.

A vector (***x,y,z***) relative to the ***z*** axis is :

$$x\mathbf{s} + y\mathbf{t} + z\mathbf{n}$$

The ONB class

```python
class ONB():
    def __init__(self) -> None:
        self.axis[0] = Vec3()
        self.axis[1] = Vec3()
        self.axis[2] = Vec3()

    def u(self):
        return self.axis[0]

    def v(self):
        return self.axis[1]

    def w(self):
        return self.axis[2]

    def local(self, val):
        if isinstance(val, Vec3):
            return self.u()*val.x() + self.v()*val.y() + self.w()*val.z()
        else:
            return self.u()*val[0] + self.v()*val[1] + self.w()*val[2]

    def build_from_w(self, vNormal):
        unit_w = Vec3.unit_vector(vNormal)
        vec_a = Vec3(1, 0, 0)
        if math.fabs(unit_w.x()) > 0.9:
            vec_a = Vec3(0, 1, 0)
        vec_v = Vec3.unit_vector(Vec3.cross_product(unit_w, vec_a))
        vec_u = Vec3.cross_product(unit_w, vec_v)

        self.axis[0] = vec_u
        self.axis[1] = vec_v
        self.axis[2] = unit_w
```

Replace the scattering

```python
class Lambertian(Material):
    def __init__(self, cAlbedo) -> None:
        super().__init__()
        self.color_albedo = rtu.Color(cAlbedo.r(), cAlbedo.g(), cAlbedo.b())

    def scattering(self, rRayIn, hHinfo):
        uvw = rtu.ONB()
        uvw.build_from_w(hHinfo.getNormal())

        scattered_direction = uvw.local(rtu.Vec3.random_cosine_hemisphere_on_z())
        scattered_ray = rtr.Ray(hHinfo.getP(), scattered_direction)
        attenuation_color = self.BRDF(rRayIn, scattered_ray, hHinfo)
        return rtu.Scatterinfo(scattered_ray, attenuation_color)

    def BRDF(self, rView, rLight, hHinfo):
        attenuation_color = rtu.Color(self.color_albedo.r(), self.color_albedo.g(), self.color_albedo.b())
        return attenuation_color
```

Final words for Image reconstruction

In ideal reconstruction, the uniform sampling is required.

For image sampling, nonuniform sampling is widely used as a trade-off between noise and aliasing.

The reconstruction techniques have been shifted towards minimizing errors.

Codes and class assignment !

- Github : RT-python-week09
  - https://github.com/KUGA-01418283-Raytracing/RT-python-week09