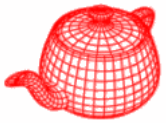
Project #1: Classes for vectors, points and rays

Due: 5:00pm 8/24

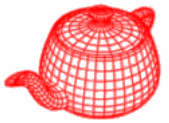
Submission: send your java sources in a zip file and send it to me.

goals

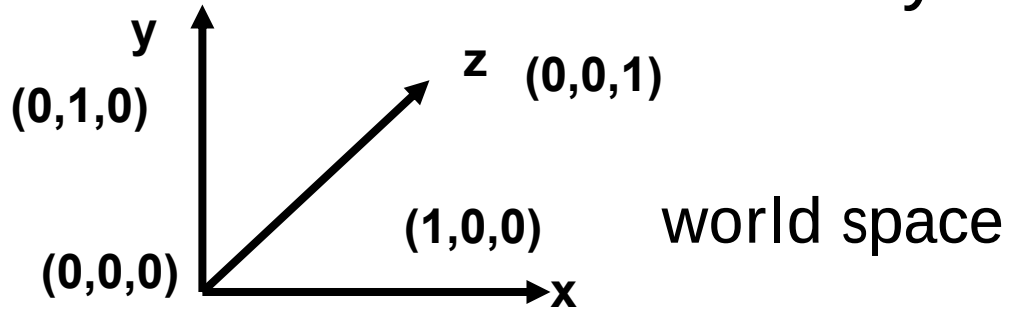


- In this project, you are asked to implement three Java classes, Vector, Point and Ray for geometric primitives.
- Your classes should support the functions listed in the following slides. You are free to design the interface as long as you support the operations.
- For each class, in addition to the required functions, you should also implement conventional member functions such as equals and toString.

Coordinate system

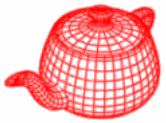


- Points, vectors and normals are represented with three floating-point coordinate values: *x, y, z* defined under a coordinate system.
- A coordinate system is defined by an origin p_o and a frame (linearly independent vectors v_i).
- A vector $v = s_1v_1 + \ldots + s_nv_n$ represents a direction, while a point $p = p_o + s_1v_1 + \ldots + s_nv_n$ represents a position. They are not freely interchangeable.
- We will use left-handed coordinate system.



world space

Vectors



```
class Vector {
  public:
      <Vector Public Methods>
      float x, y, z;
}
```

no need to use selector (getX) and mutator (setX)
because the design gains nothing and adds bulk to its usage

Provided operations: **`Vector u, v; float a;`**

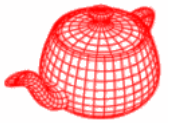
`v+u, v-u`

`-v`

`(v==u)`

`a*v, v/a`

Dot and cross product



`Dot(v, u)`
`AbsDot(v, u)`

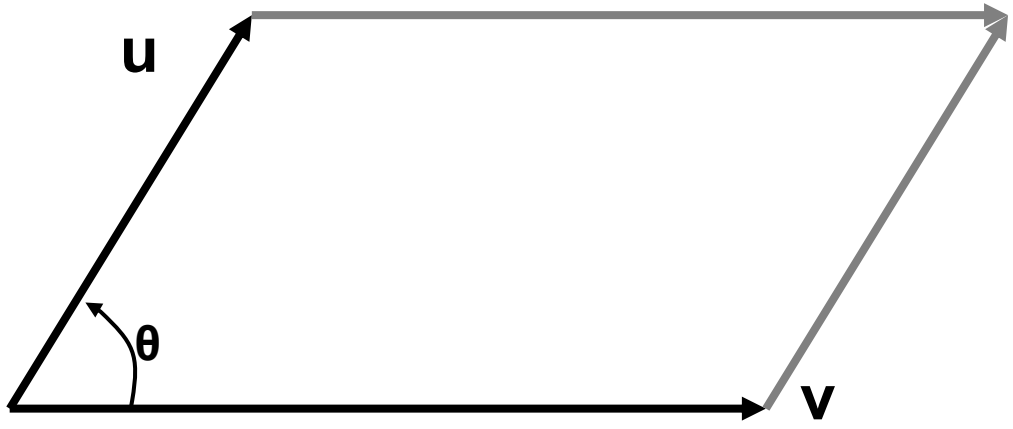
$$v \cdot u = \|v\|\|u\| \cos\theta$$

`Cross(v, u)`

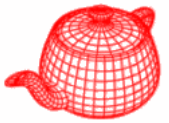
$$\|v \times u\| = \|v\|\|u\| \sin\theta$$

Vectors v, u, v×u form a frame

$$(v \times u)_x = v_y u_z - v_z u_y$$
$$(v \times u)_y = v_z u_x - v_x u_z$$
$$(v \times u)_z = v_x u_y - v_y u_z$$



Normalization



`a=LengthSquared(v)`

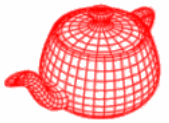
`a=Length(v)`

`u=Normalize(v)` *return a vector, does not normalize in place*

Take normalize as an example, you can implement it in the following two forms (there are other possibilities):

1. u = v.normalize(); // where normalize is a member function
   // I personally prefer this one
2. u = Vector.normalize(v); // where v is a static function

Points



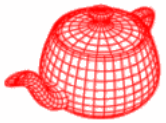
Points are different from vectors; given a coordinate system (p_0,v_1,v_2,v_3), a point p and a vector v with the same (x,y,z) essentially means

$p=(x,y,z,1)[v_1\ v_2\ v_3\ p_0]^T$

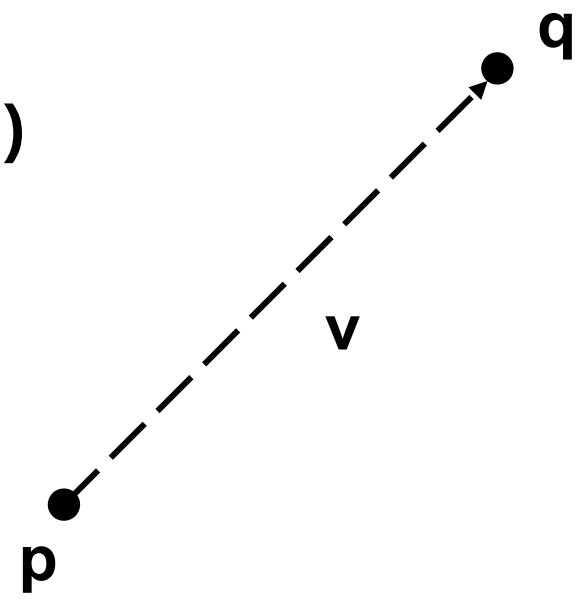
$v=(x,y,z,0)[v_1\ v_2\ v_3\ p_0]^T$

```
Vector(Point p); //converts point to vector
```

Operations for points



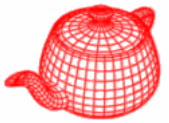
```
Vector v; Point p, q, r; float a;

q=p+v; // q=p.add(v)
q=p-v;
v=q-p;

r=p+q;
a*p; p/a;
```



(This is only for the operation αp+βq.)

```
Distance(p,q);
DistanceSquared(p,q);
```

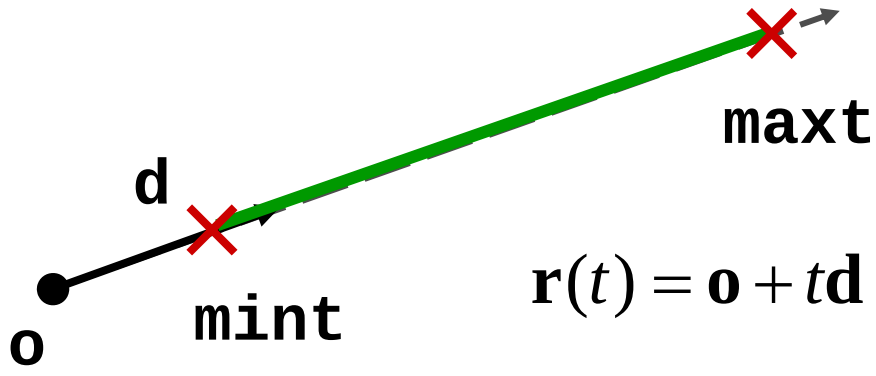
Rays



```
class Ray {
public:
  Point o;
  Vector d;
  float mint, maxt;
  int depth;
};
```

Initialized as a small and a large number respectively

(how many times the ray has bounced, ignore for now)




```
Ray r(o, d);
Point p=r.at(t);
```

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d} \qquad 0 \le t \le \infty$$