INTRODUCTION

The **P3F** (P3D file Format) is designed as a minimal scene description language for the scenes used to benchmark the ray tracers developed by the students in P3D Course. It is based on the NFF (Neutral File Format) language developed by Eric Haines.

P3F is meant to describe the geometry and basic surface characteristics of objects, the placement of lights, and the viewing frustum for the eye and has minimal support for lighting and shading. However, P3F extends NFF, by providing parameterization to support additional geometries like meshes and axis-aligned boxes, different types of luminaries, ray accelerators, number of random samples per pixel, specular color in the material, environmental cubemaps and an extended camera with depth-of-field (DOF) effect.

Presently the following entities are supported:

- Accelerator type
- Number of random samples per pixel
- Viewing camera with simple perspective frustum and thin lens
- A background color description
- A Environmental cubemap (skybox)
- Punctual and Quad light source type descriptions
- Material shading surface properties
- Geometric objects: plane, Polygon, polygonal patch, sphere, box, cylinder/cone and triangle mesh descriptions.

Files are output as lines of text. For each entity, the first field defines its type. The rest of the line and possibly other lines contain further information about the entity. Entities include:

"accel" – acceleration data structure
"spp" – number of random samples per pixel
"camera" - viewing camera
"bclr" - background color
"env" – environmental cubemap (skybox)
"light" - luminaries (light sources)
"mat" - object material properties
"c" - cone or cylinder primitive
"s" - sphere primitive
"pl" - plane primitive
"box" - box primitive
"p" - polygon primitive
"mesh" – mesh of triangles
"pp" - polygonal patch primitive

These are explained in depth below.

DESCRIPTION

Comment. Description:
    "#" [string]

Format:
    # [string]

    As soon as a "#" character is detected, the rest of the line is considered a comment.

Accelerator type. It contains the type of acceleration data structure.0 means no acceleration; 1 means regular Grid-based and 2 means BVH (Bounding Volume Hierarchy)-based. Description:

"accel" type_of_acelerator

type_of_acelerator can be: {“none”, “grid”, “bvh”}

Format:
accel [string]

Number of random samples per pixel. It contains the number of random samples per pixel. **The special case with 0 value means just one fixed (not-random) sample per pixel**. Description:

"spp" number_of_random_samples

Format:
spp %d

Viewing camera. Description:

    "camera"
    "eye" Fx Fy Fz
    "at" Ax Ay Az
    "up" Ux Uy Uz
    "angle" angle
    "hither" hither
    "resolution" xres yres
    "aperture" aperture_ratio
    "focal" focal_ratio

Format:

```
    camera
    eye %f %f %f
    at %f %f %f
    up %f %f %f
    angle %f
    hither %f
    resolution %d %d
    aperture %f
    focal %f
```

The parameters are:

Eye: the eye location in XYZ.
At: a position to be at the center of the image, in XYZ world coordinates. A.k.a. "lookat".
Up: a vector defining which direction is up, as an XYZ vector.
Angle: in degrees; it defines the vertical fov (fovy.
Hither: distance of the hither plane (if any) from the eye. Mostly needed for hidden surface algorithms.
Resolution: in pixels, in x and in y
Aperture: lens aperture expressed as multiple of pixel size
Focal: ratio between the focal distance and the viewplane distance.

Note that no assumptions are made about normalizing the data (e.g. the eye-at distance does not have to be 1). Also, vectors are not required to be perpendicular to each other.

Some viewing parameters are always the same:
Yon is "at infinity."
Aspect ratio is 1.0.

Background color. A color is simply RGB with values between 0 and 1:

```
    "bclr" R G B
```

Format:

```
    bclr %f %f %f
```

If no background color is set, assume RGB = {0,0,0}.

Environmental cubemap (skybox). Directory name containing the 6 maps of the skybox:

```
    "env" cubemap_dir
```

Format:

```
    env %s
```

Light. The light source can be of two types: punctual (point) and quad. The punctual light is defined by a XYZ position (pos) and a color emission RGB. The quad light is defined by a XYZ position (pos), two additional points (v1 and v2), a color emission RGB and the number of samples to be used in light sampling. The points pos, v1 and v2 form a rectangular area defined by two side vectors e1 = v1 – pos and e2 = v1 – pos.

Description of a **punctual** luminary:

"light" "punctual" X Y Z R G B

Format:
light punctual %f %f %f %f %f %f

Description of a **quad** area luminary:

"light" "quad" X Y Z R G B V1.X V1.Y V1.Z V2.X V2.Y V2.Z num_samples

Format:
light quad %f %f %f %f %f %f %f %f %f %f %f %f %d

Material shading parameters. Description:

"mat" red green blue Kd red green blue Ks Shine T index_of_refraction

Format:
mat %f %f %f %f %f %f %f %f %f %f %f

Each RGB channel ranges from 0.0 to 1.0.

The first RGB color regards the diffuse color. The second RGB specifies the specular color. Kd is the diffuse component, Ks the specular, Shine is the Phong cosine power for highlights and T is transmittance (0 or 1). Usually, 0 <= Kd <= 1 and 0 <= Ks <= 1, though it is not required that Kd + Ks == 1.
If T = 0, the object is opaque.
If T = 1, then the object is transmissive, and it should be used the Fresnel equations to calculate both reflectance and transmittance (the fractions of the contributions of the reflecting and transmitting rays).

The material is used to shade the objects following it until a new material is assigned.

Sphere. A sphere is defined by a radius and center position:

```
"s" center.x center.y center.z radius
```

Format:

```
s %f %f %f %f
```

If the radius is negative, then only the sphere's inside is visible (objects are normally considered one sided, with the outside visible). Currently none of the test scenes make use of negative radii.

Box. A box is defined by the coordinates of the two opposite points (min and max):

```
"box" min.x min.y min.z max.x max.y max.z
```

Format:

```
box %f %f %f %f %f %f
```

Plane. A plane is defined by the coordinates of three points:

```
"pl" p1.x p1.y p1.z p2.x p2.y p2.z p3.x p3.y p3.z
```

Format:

```
pl %f %f %f %f %f %f %f %f %f
```

Polygon. A polygon is defined by a set of vertices. With these databases, a polygon is defined to have all points coplanar. A polygon has only one side, with the order of the vertices being counterclockwise as you face the polygon (right-handed coordinate system). The first two edges must form a non-zero convex angle, so that the normal and side visibility can be determined by using just the first three vertices. Description:

```
"p" total_vertices
vert1.x vert1.y vert1.z
[etc. for total_vertices vertices]
```

Format:

```
p %d
[ %f %f %f ] <-- for total_vertices vertices
```

Mesh. A mesh is defined by a set of vertices and an indexed faces (triangles) set. With these databases, each triangle has the same properties described above. The indexed face set starts with 1 (first vertex of the vertices set) or -1 (the last vertex of the vertices set). Description:

```
"mesh" total_vertices total_triangles
vert1.x vert1.y vert1.z
[etc. for total_vertices vertices]
index1 index2 index3
[etc. for total_triangles triangles]
```

Format:

```
p %d %d
[ %f %f %f ] <-- for total_vertices vertices
[ %d %d %d ] <-- for total_triangles triangles
```

Polygonal patch. A patch is defined by a set of vertices and their normals.

With these databases, a patch is defined to have all points coplanar. A patch has only one side, with the order of the vertices being counterclockwise as you face the patch (right-handed coordinate system).

The first two edges must form a non-zero convex angle, so that the normal and side visibility can be determined. Description:

```
"pp" total_vertices
vert1.x vert1.y vert1.z norm1.x norm1.y norm1.z
[etc. for total_vertices vertices]
```

Format:

```
pp %d
[ %f %f %f %f %f %f ] <-- for total_vertices vertices
```

Cylinder or cone. A cylinder is defined as having a radius and an axis defined by two points, which also define the top and bottom edge of the cylinder. A cone is defined similarly, the difference being that the apex and base radii are different. The apex radius is defined as being smaller than the base radius. Note that the surface exists without endcaps. The cone or cylinder description:

```
"c"
base.x base.y base.z base_radius
```

apex.x apex.y apex.z apex_radius

Format:

```
c
%f %f %f %f
%f %f %f %f
```

A negative value for both radii means that only the inside of the object is visible (objects are normally considered one sided, with the outside visible). Note that the base and apex cannot be coincident for a cylinder or cone. Making them coincident could be used to define endcaps, but none of the test scenes currently make use of this definition.