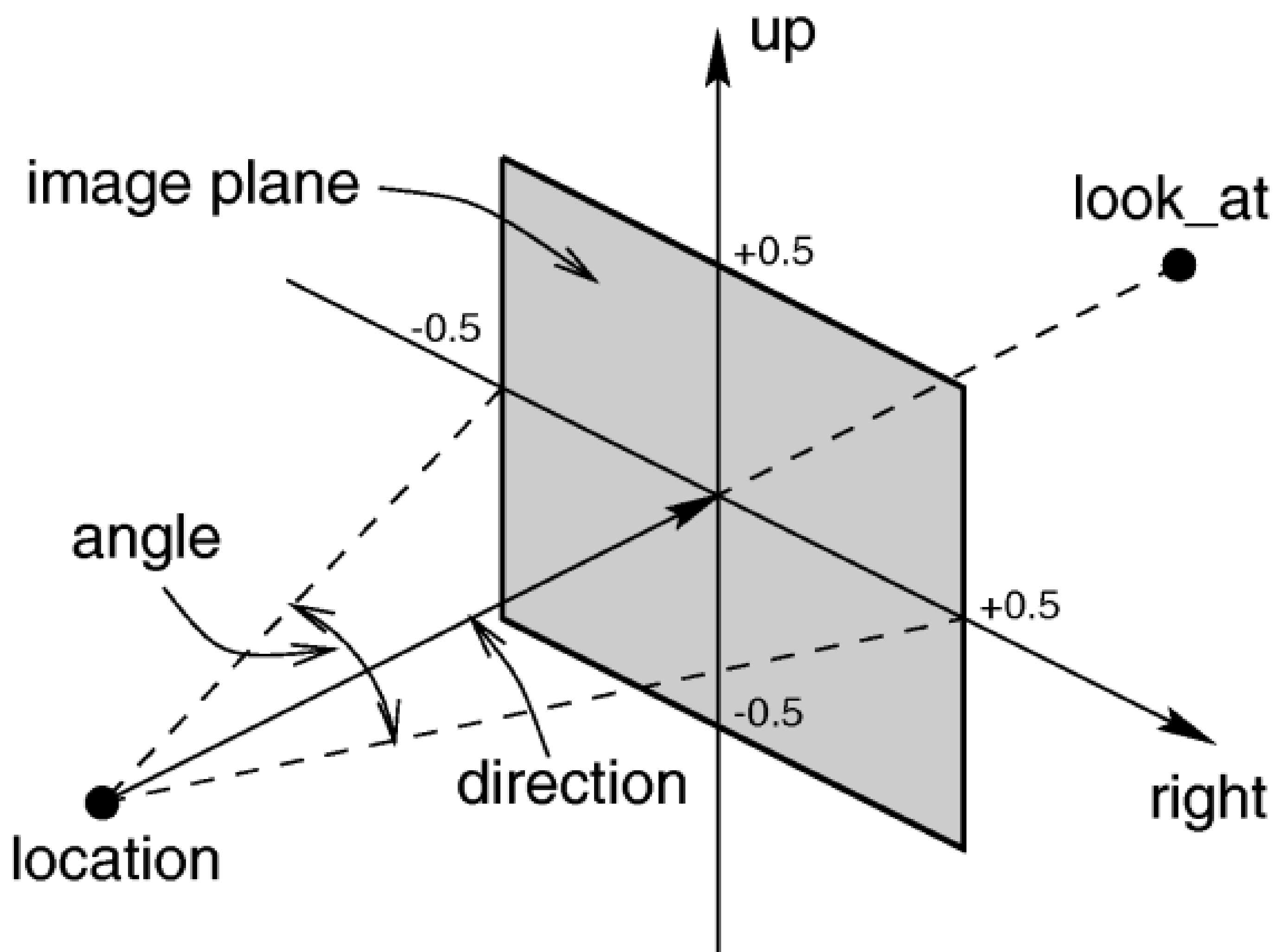
camera {

The POV-Ray camera has ten different models, each of which uses a different projection method to project the scene onto your screen.
Regardless of the projection type all cameras use the location, right, up, direction, and keywords to determine the location and orientation of the camera. The type keywords and these four vectors fully define the camera. All other camera modifiers adjust how the camera does its job.
The meaning of these vectors and other modifiers differ with the projection type used. A more detailed explanation of the camera types follows later. In the sub-sections which follows, we explain how to place and orient the camera by the use of these four vectors and the sky and look_at modifiers. You may wish to refer to the illustration of the perspective camera below as you read about these vectors.



-/ types of projection \-

Default : perspective
Valid values : perspective | orthographic | fisheye | ultra_wide_angle | omnimax | panoramic | cylinder 1-2-3-4
Example :

```
camera {
  omnimax
  location <0, 2, -20>
  right <4/3, 0, 0>
  up <0, 1, 0>
  direction <0, 0, 1>
  look_at <0, 2, 10>
}
```

perspective

The perspective specifies the default perspective camera which simulates the classic pinhole camera. The (horizontal) viewing angle is either determined by the ratio between the length of the direction vector and the length of the right vector or by the optional keyword angle, which is the preferred way. The viewing angle has to be larger than 0 degrees and smaller than 180 degrees. See the figure in "Placing the Camera" for the geometry of the perspective camera.

orthographic

This projection uses parallel camera rays to create an image of the scene. The size of the image is determined by the lengths of the right and up vectors.

If you add the orthographic keyword after all other parameters of a perspective camera you'll get an orthographic view with the same image area, i.e. the size of the image is the same. In this case you needn't specify the lengths of the right and up vector because they'll be calculated automatically. You should be aware though that the visible parts of the scene change when switching from perspective to orthographic view. As long as all objects of interest are near the look_at point they'll be still visible if the orthographic camera is used. Objects farther away may get out of view while nearer objects will stay in view.

fisheye

This is a spherical projection. The viewing angle is specified by the angle keyword. An angle of 180 degrees creates the "standard" fisheye while an angle of 360 degrees creates a super-fisheye ("I-see-everything-view"). If you use this projection you should get a circular image. If this isn't the case, i.e. you get an elliptical image, you should read "Aspect Ratio".

ultra_wide_angle

This projection is somewhat similar to the fisheye but it projects the image onto a rectangle instead of a circle.
The viewing angle can be specified using the angle keyword.

omnimax

The omnimax projection is a 180 degrees fisheye that has a reduced viewing angle in the vertical direction. In reality this projection is used to make movies that can be viewed in the dome-like Omnimax theaters. The image will look somewhat elliptical. The angle keyword isn't used with this projection.

panoramic

This projection is called "cylindrical equirectangular projection". It overcomes the degeneration problem of the perspective projection if the viewing angle approaches 180 degrees. It uses a type of cylindrical projection to be able to use viewing angles larger than 180 degrees with a tolerable lateral-stretching distortion. The angle keyword is used to determine the viewing angle.

cylinder

Using this projection the scene is projected onto a cylinder. There are four different types of cylindrical projections depending on the orientation of the cylinder and the position of the viewpoint. A float value in the range 1 to 4 must follow the cylinder keyword.
The viewing angle and the length of the up or right vector determine the dimensions of the camera and the visible image. The camera to use is specified by a number. The types are:

```
1   vertical cylinder, fixed viewpoint
2   horizontal cylinder, fixed viewpoint
3   vertical cylinder, viewpoint moves along the cylinder's axis
4    horizontal cylinder, viewpoint moves along the cylinder's axis
```

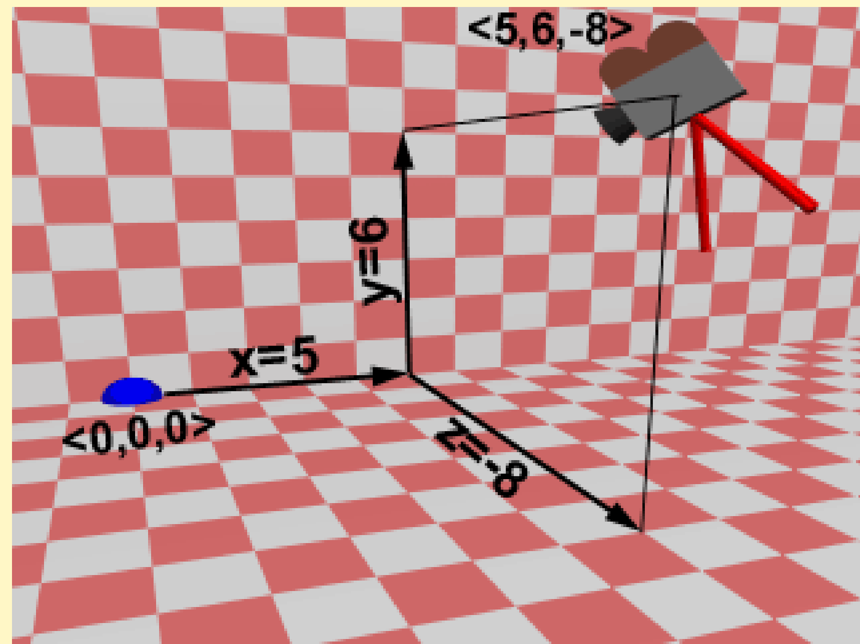
You should note that the vista buffer can only be used with the perspective and orthographic camera.

location

Default : <0, 0 , 0>
Valid values : any vector

The location is simply the x, y, z coordinates of the camera. The camera can be located anywhere in the ray-tracing universe.

```
camera {
        location < 5 , 6 ,-8>
        look_at  <  0 ,  0 ,  0>
        }
```



angle

The angle keyword followed by a float expression specifies the (horizontal) viewing angle in degrees of the camera used. Even though it is possible to use the direction vector to determine the viewing angle for the perspective camera it is much easier to use the angle keyword.

When you specify the angle, POV-Ray adjusts the length of the direction vector accordingly. The formula used is direction_length = 0.5 *right_length / tan(angle / 2) where right_length is the length of the right vector. You should therefore specify the direction and right vectors before the angle keyword. The right vector is explained in the next section.

There is no limitation to the viewing angle except for the perspective projection. If you choose viewing angles larger than 360 degrees you'll see repeated images of the scene (the way the repetition takes place depends on the camera). This might be useful for special effects.

direction

Default : <0 , 0 , 1>
Typical values : any vector
Example :

```
camera {
       location  <5 , 2 , -3>
       direction <0 , 0 ,  1>
       look_at   <0 , 0 ,  0>
       }
```

You will probably not need to explicitly specify or change the camera direction vector but it is described here in case you do. It tells POV-Ray the initial direction to point the camera before moving it with the look_at or rotate vectors (the default value is direction<0,0,1>). It may also be used to control the (horizontal) field of view with some types of projection. The length of the vector determines the distance of the viewing plane from the camera's location. A shorter direction vector gives a wider view while a longer vector zooms in for close-ups. In early versions of POV-Ray, this was the only way to adjust field of view. However zooming should now be done using the easier to use angle keyword.

If you are using the ultra_wide_angle, panoramic, or cylindrical projection you should use a unit length direction vector to avoid strange results.

The length of the direction vector doesn't matter when using the orthographic, fisheye, or omnimax projection types.

aperture

Default : 0
Typical values : 0.1-3
Example :

```
camera {
        location      <10 , 10 , -10>
        blur_samples  100
        aperture      1
        variance      1/256
        confidence    0.99
        focal_point   <0  , 0 ,   0>
        look_at       <0  , 0 ,   0>
}
```

To turn on focal blur, you must specify the aperture keyword followed by a float value which determines the depth of the sharpness zone. Large apertures give a lot of blurring, while narrow apertures will give a wide zone of sharpness. Note that, while this behaves as a real camera does, the values for aperture are purely arbitrary and are not related to f-stops

blur_samples

Default : 0
Typical values : 10-120
Example :

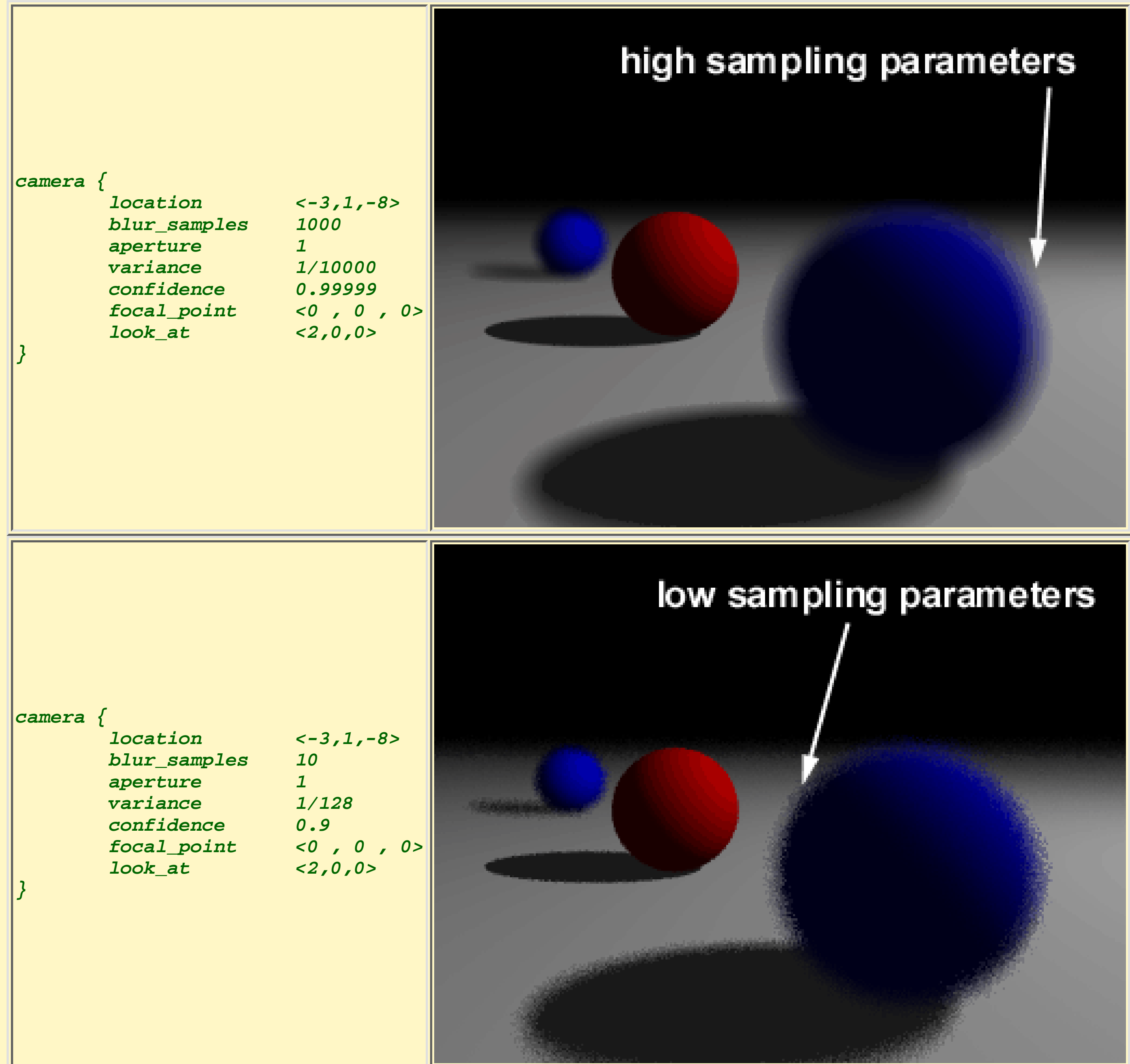
```
camera {
        location      <10 , 10 , -10>
        blur_samples  100
        aperture      1
        variance      1/256
        confidence    0.99
        focal_point   <0  , 0 ,   0>
        look_at       <0  , 0 ,   0>
}
```

You must also specify the blur_samples keyword followed by an integer value specifying the maximum number of rays to use for each pixel.
More rays give a smoother appearance but is slower. By default no focal blur is used, i. e. the default aperture is 0 and the default number of samples is 0.

See confidence for example

confidence

Default : 0.9
Typical values : 0.9 -> 0.9999999 The higher, the slower (and the more accurate, but no more than 1)
Example :




```
camera {
        location        <-3,1,-8>
        blur_samples    1000
        aperture        1
        variance        1/10000
        confidence      0.99999
        focal_point     <0 , 0 , 0>
        look_at         <2,0,0>
}
```

```
camera {
        location        <-3,1,-8>
        blur_samples    10
        aperture        1
        variance        1/128
        confidence      0.9
        focal_point     <0 , 0 , 0>
        look_at         <2,0,0>
}
```

Although blur_samples specifies the maximum number of samples, there is an adaptive mechanism that stops shooting rays when a certain degree of confidence has been reached. At that point, shooting more rays would not result in a significant change. The confidence and variance keywords are followed by float values to control the adaptive function. The confidence value is used to determine when the samples seem to be close enough to the correct color. The variance value specifies an acceptable tolerance on the variance of the samples taken so far. In other words, the process of shooting sample rays is terminated when the estimated color value is very likely (as controlled by the confidence probability) near the real color value.

Since the confidence is a probability its values can range from 0 to 1 (the default is 0.9, i. e. 90%). The value for the variance should be in the range of the smallest displayable color difference (the default is 1/128).

Larger confidence values will lead to more samples, slower traces and better images. The same holds for smaller variance thresholds.

variance

Default : 1/128
Typical values : 1/128 -> 1/1000 The lower, the slower (and the more accurate, but always more than 0)
Example :

```
camera {
        location      <10 , 10 , -10>
        blur_samples  100
        aperture      1
        variance      1/256
        confidence    0.99
        focal_point   <0  , 0 ,   0>
        look_at       <0  , 0 ,   0>
        }
```

Although blur_samples specifies the maximum number of samples, there is an adaptive mechanism that stops shooting rays when a certain degree of confidence has been reached. At that point, shooting more rays would not result in a significant change. The confidence and variance keywords are followed by float values to control the adaptive function. The confidence value is used to determine when the samples seem to be close enough to the correct color. The variance value specifies an acceptable tolerance on the variance of the samples taken so far. In other words, the process of shooting sample rays is terminated when the estimated color value is very likely (as controlled by the confidence probability) near the real color value.

Since the confidence is a probability its values can range from 0 to 1 (the default is 0.9, i. e. 90%). The value for the variance should be in the range of the smallest displayable color difference (the default is 1/128).

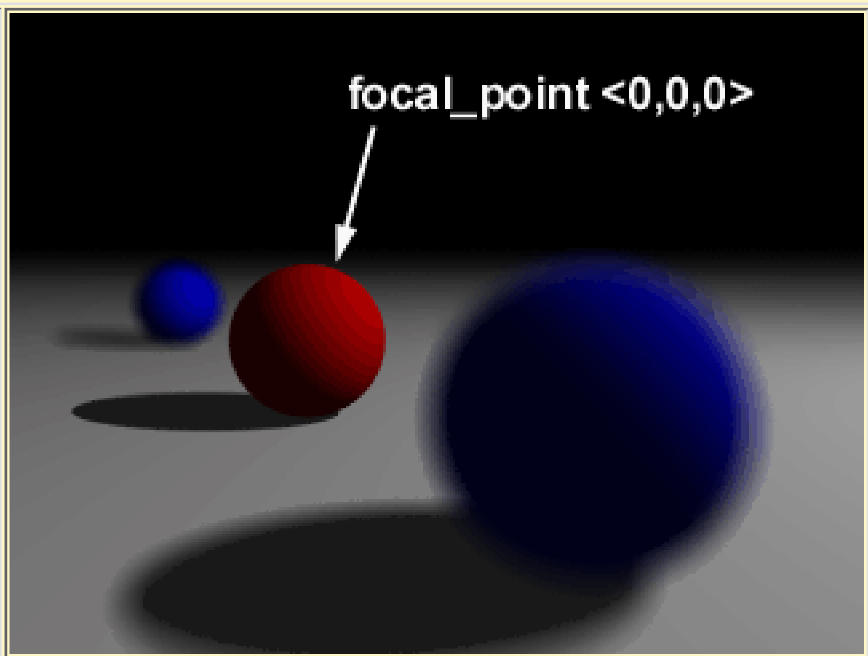
Larger confidence values will lead to more samples, slower traces and better images. The same holds for smaller variance thresholds.

See confidence for example.

focal_point

Default : < 0 , 0 , 0>
Typical values : any vector
Example :

```
camera {
        location        <-3,1,-8>
        blur_samples    1000
        aperture        1
        varnciae        1/10000
        confidence      0.99999
        focal_point     <0 , 0 , 0>
        look_at         <2,0,0>
}
```

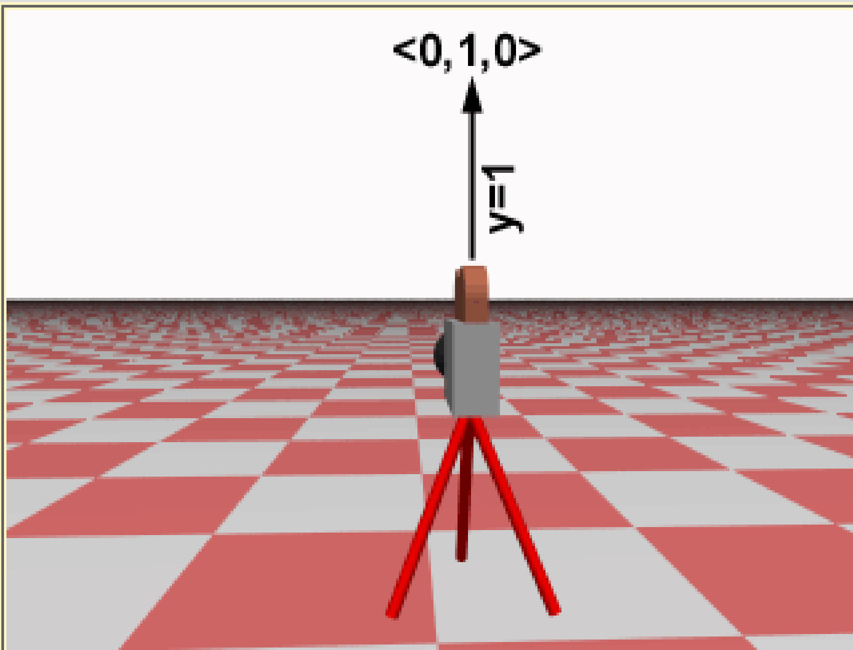


The center of the zone of sharpness is specified by the focal_point vector. Objects close to this point are in focus and those farther from that point are more blurred.

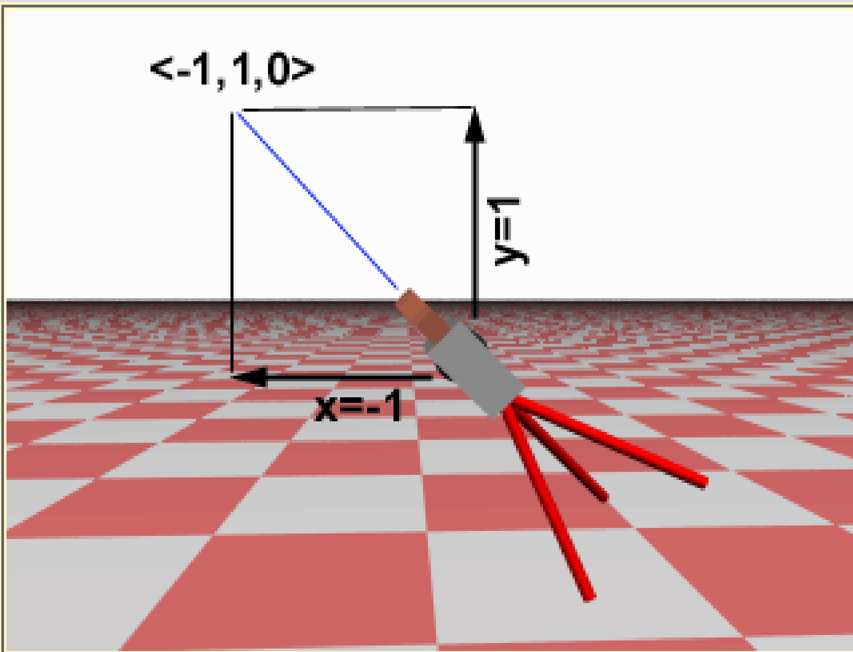
sky

Default : <0 , 1 , 0>
Valid values : any vector
Example :

```
camera {
        location <0,0,-10>
        sky    <0,1,0>
        look_at <0,0,0>
        }
```




```
camera {
        location <0,0,-10>
        sky    <-1,1,0>
        look_at <0,0,0>
        }
```



Normally POV-Ray pans left or right by rotating about the y-axis until it lines up with the look_at point and then tilts straight up or down until the point is met exactly. However you may want to slant the camera sideways like an airplane making a banked turn. You may change the tilt of the camera using the sky vector. For example:

This tells POV-Ray to roll the camera until the top of the camera is in line with the sky vector. Imagine that the sky vector is an antenna pointing out of the top of the camera. Then it uses the sky vector as the axis of rotation left or right and then to tilt up or down in line with the sky until pointing at the look_at point. In effect you're telling POV-Ray to assume that the sky isn't straight up. Note that the sky vector must appear before the look_at vector.

The sky vector does nothing on its own. It only modifies the way the look_at vector turns the camera.

up

Default : <0 , 1 , 0>
Valid values : any vector
Example :

```
camera {
        location < 3 , 5 , -10 >
        up       < 0 , 1 ,   0 > | y
        look_at  < 0 , 2 ,   1 >
        }
```

right

Default : <1.33 , 0 , 0>
Valid values : any vector
Example :

```
camera {
        location <3 , 5 , -10>
        right    <1 , 0 ,   0> | x*4/3
        look_at  <0 , 2 ,   1>
        }
```

In the default perspective camera, these two vectors also define the initial plane of the view screen before moving it with the look_at or rotate vectors. The length of the right vector (together with the direction vector) may also be used to control the (horizontal) field of view with some types of projection. The look_at modifier changes both up and right so you should always specify them before look_at. Also the angle calculation depends on the right vector so right should precede it.

Most camera types treat the up and right vectors the same as the perspective type. However several make special use of them. In the orthographic projection: The lengths of the up and right vectors set the size of the viewing window regardless of the direction vector length, which is not used by the orthographic camera.

When using cylindrical projection: types 1 and 3, the axis of the cylinder lies along the up vector and the width is determined by the length of right vector or it may be overridden with the angle vector.
In type 3 the up vector determines how many units high the image is. For example if you have up 4*y on a camera at the origin. Only points from y=2 to y=-2 are visible. All viewing rays are perpendicular to the y-axis.
For type 2 and 4, the cylinder lies along the right vector. Viewing rays for type 4 are perpendicular to the right vector.

Note that the up, right, and direction vectors should always remain perpendicular to each other or the image will be distorted. If this is not the case a warning message will be printed. The vista buffer will not work for non-perpendicular camera vectors. If you specify the 3 vectors as initially perpendicular and do not explicitly re-specify the after any look_at or rotate vectors, the everything will work fine.

Aspect Ratio

Together the up and right vectors define the aspect ratio (height to width ratio) of the resulting image. The default values up<0,1,0> and right<1.33,0,0> result in an aspect ratio of 4 to 3. This is the aspect ratio of a typical computer monitor. If you wanted a tall skinny image or a short wide panoramic image or a perfectly square image you should adjust the up and right vectors to the appropriate proportions.

Most computer video modes and graphics printers use perfectly square pixels. For example Macintosh displays and IBM SVGA modes 640x480, 800x600 and 1024x768 all use square pixels. When your intended viewing method uses square pixels then the width and height you set with the Width and Height options or +W or +H switches should also have the same ratio as the up and right vectors. Note that 640/480 = 4/3 so the ratio is proper for this square pixel mode.

Not all display modes use square pixels however. For example IBM VGA mode 320x200 and Amiga 320x400 modes do not use square pixels. These two modes still produce a 4/3 aspect ratio image. Therefore images intended to be viewed on such hardware should still use 4/3 ratio on their up and right vectors but the pixel settings will not be 4/3.

Example :

```
camera {
          location <3 , 5 , -10>
          up       <0 , 1 ,   0>
          right    <1 , 0 ,   0>
          look_at  <0 , 2 ,   1>
}
```

This specifies a perfectly square image. On a square pixel display like SVGA you would use pixel settings such as +W480 +H480 or +W600 +H600.
However on the non-square pixel Amiga 320x400 mode you would want to use values of +W240 +H400 to render a square image.

The bottom line issue is this: the up and right vectors should specify the artist's intended aspect ratio for the image and the pixel settings should be adjusted to that same ratio for square pixels and to an adjusted pixel resolution for non-square pixels. The up and right vectors should not be adjusted based on non-square pixels.

Handedness

The right vector also describes the direction to the right of the camera. It tells POV-Ray where the right side of your screen is. The sign of the right vector can be used to determine the handedness of the coordinate system in use. The default value is: right<1.33,0,0>. This means that the +x-direction is to the right. It is called a left-handed system because you can use your left hand to keep track of the axes. Hold out your left hand with your palm facing to your right. Stick your thumb up. Point straight ahead with your index finger. Point your other fingers to the right. Your bent fingers are pointing to the +x-direction. Your thumb now points into +y-direction. Your index finger points into the +z-direction.

To use a right-handed coordinate system, as is popular in some CAD programs and other ray-tracers, make the same shape using your right hand. Your thumb still points up in the +y-direction and your index finger still points forward in the +z-direction but your other fingers now say the +x-direction is to the left. That means that the right side of your screen is now in the -x-direction. To tell POV-Ray to act like this you can use a negative x value in the right vector such as: right<-1.33,0,0>. Since having x values increasing to the left doesn't make much sense on a 2D screen you now rotate the whole thing 180 degrees around by using a positive z value in your camera's location. You end up with something like this.

```
camera {
        location <    0 , 0 , 10>
        up       <    0 , 1 ,  0>
        right    <-1.33 , 0 ,  0>
        look_at  <    0 , 0 ,  0>
 }
```

Now when you do your ray-tracer's aerobics, as explained in the section "Understanding POV-Ray's Coordinate System", you use your right hand to determine the direction of rotations.

In a two dimensional grid, x is always to the right and y is up. The two versions of handedness arise from the question of whether z points into the screen or out of it and which axis in your computer model relates to up in the real world.

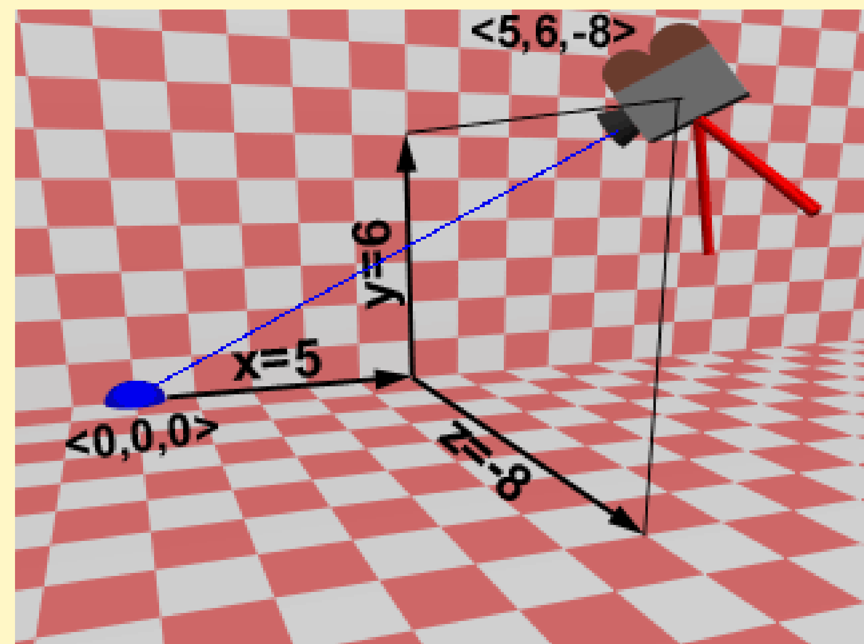
Architectural CAD systems, like AutoCAD, tend to use the God's Eye orientation that the z-axis is the elevation and is the model's up direction.
This approach makes sense if you're an architect looking at a building blueprint on a computer screen. z means up, and it increases towards you, with x and y still across and up the screen. This is the basic right handed system.

Stand alone rendering systems, like POV-Ray, tend to consider you as a participant. You're looking at the screen as if you were a photographer standing in the scene. The up direction in the model is now y, the same as up in the real world and x is still to the right, so z must be depth, which increases away from you into the screen. This is the basic left handed system.

look_at

Valid values : any vector
Example :

```
camera {
            location <5,6,-8>
            look_at <0,0,1>
            }
```



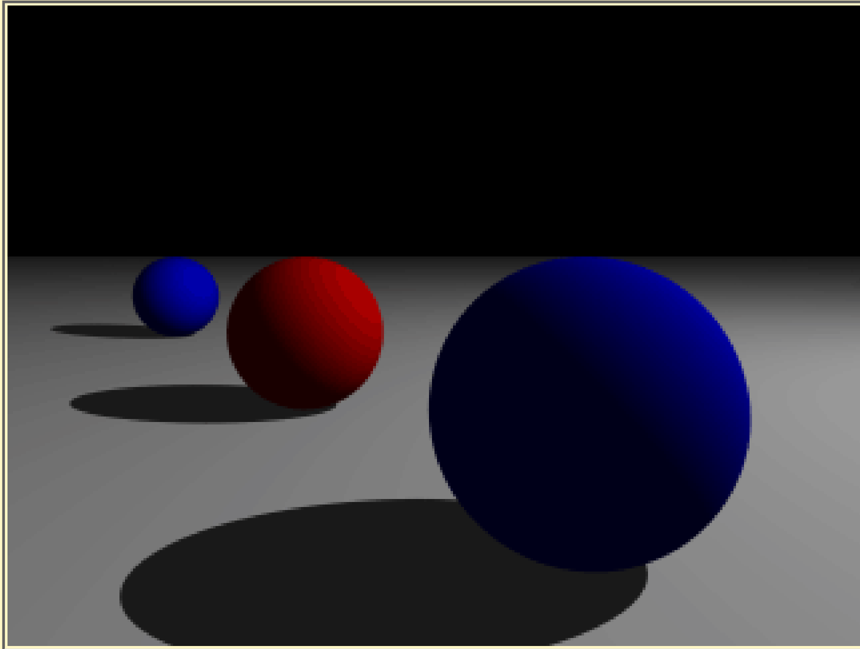
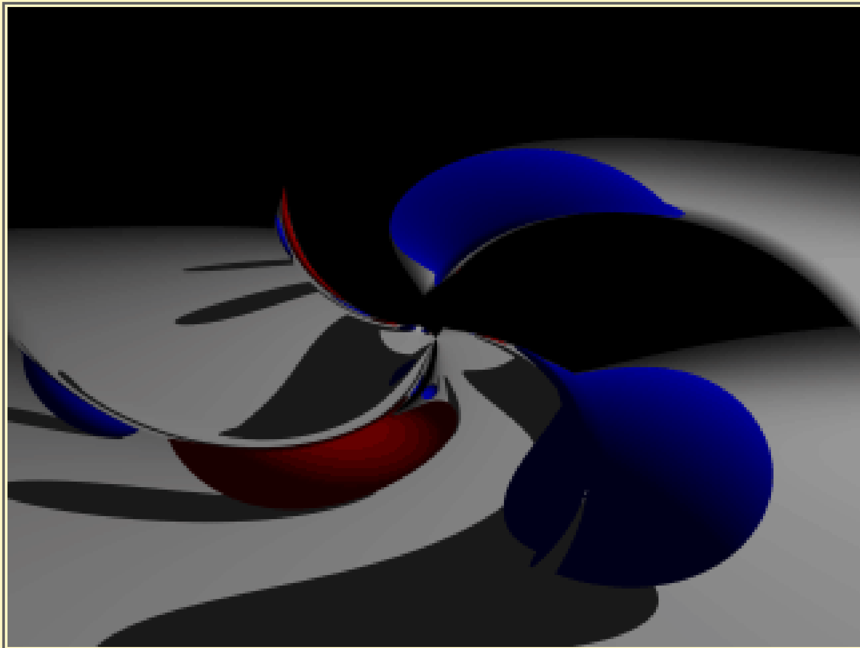
The look_at vector tells POV-Ray to pan and tilt the camera until it is looking at the specified x, y, z coordinates. By default the camera looks at a point one unit in the z-direction from the location.

The look_at modifier should almost always be the last item in the camera statement. If other camera items are placed after the look_at vector then the camera may not continue to look at the specified point.

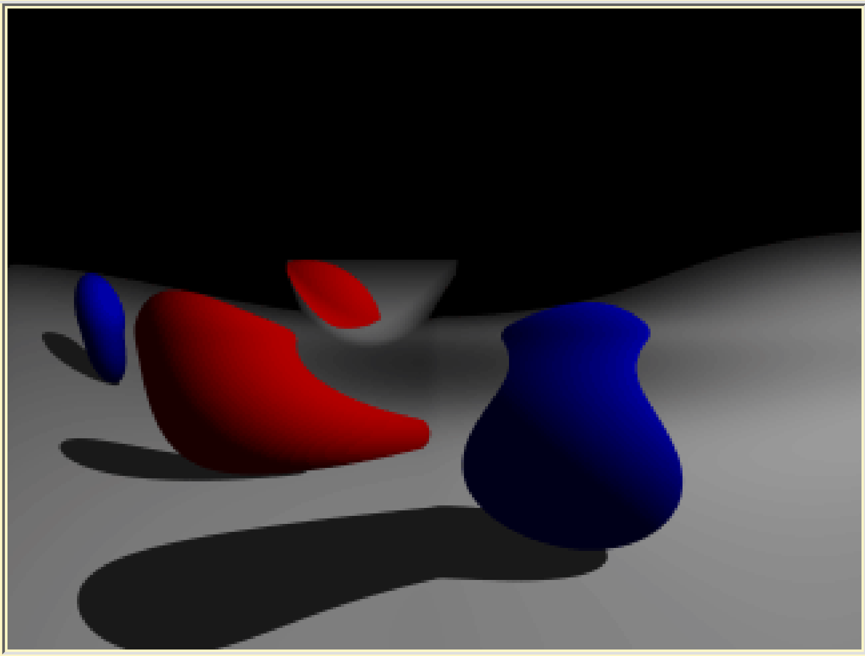
normal {

Valid values :
agate | average(+map) | boxed | bozo | brick | bumps | checker | crackle | cylindrical | density_file(+file) | dents | gradient | granite | hexagon | leopard | mandel(+val) | marble | onion | planar | quilted | radial | ripples | spherical | spiral1(+val) | spiral2(+val) | spotted | waves | wood | wrinkles

The optional normal may be used to assign a normal pattern to the camera. For example:

Original picture	
camera { location <-3,1,-8> look_at <2,0,0> normal { spiral1 2 } }	

```
camera {
        location <-3,1,-8>
        look_at  <2,0,0>
        normal { bozo scale 0.5}
        }
```



All camera rays will be perturbed using this pattern. The image will be distorted as though you were looking through bumpy glass or seeing a reflection off of a bumpy surface. This lets you create special effects. See the animated scene camera2.pov for an example. See "Normal" for information on normal patterns.