

Blender - Python API

#6



Serdar ARITAN

Department of Computer Graphics
Hacettepe University, Ankara, Turkey



Blender/Python API

Camera Settings

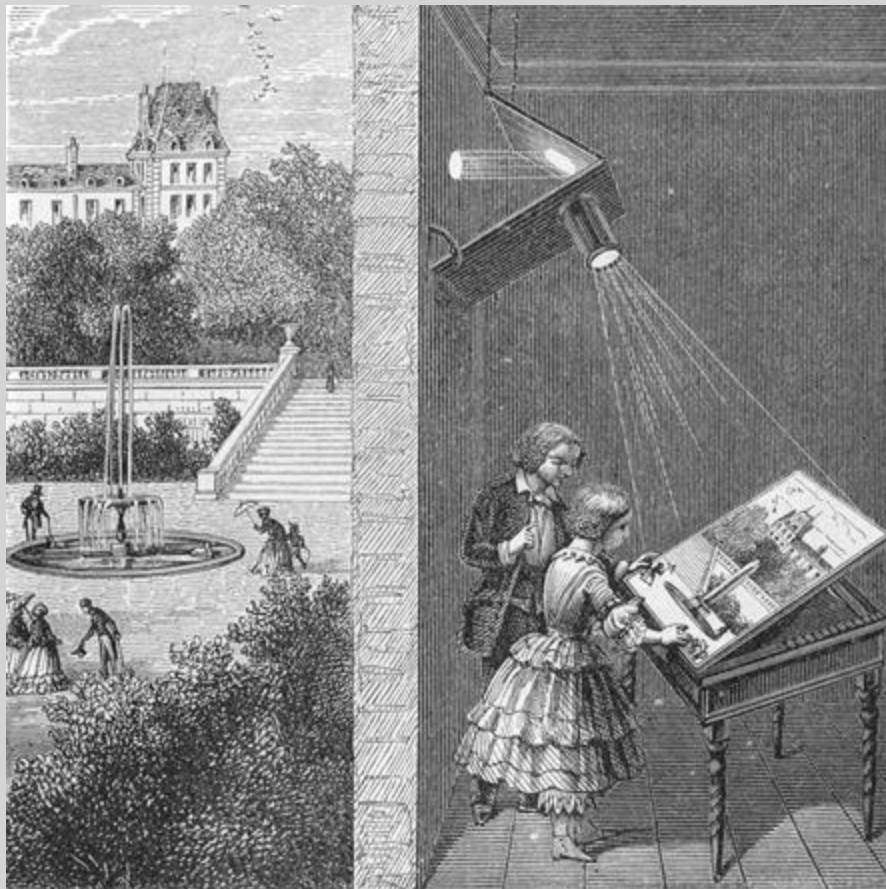
Cameras

By default the Scene has one Camera which is positioned to capture an image of the Cube Object. What the Camera sees and what is captured as an image is called Camera View. You can see Camera View in the 3D Viewport Editor by pressing **Num Pad 0** on the Keyboard. To return to User Perspective View, press **Num Pad 5** (User Orthographic View) then **Num Pad 5** again for User Perspective View



Camera obscura, ancestor of the photographic camera. The Latin name means “dark chamber,” and the earliest versions, dating to antiquity, consisted of small darkened rooms with light admitted through a single tiny hole.

Blender/Python API Camera Settings

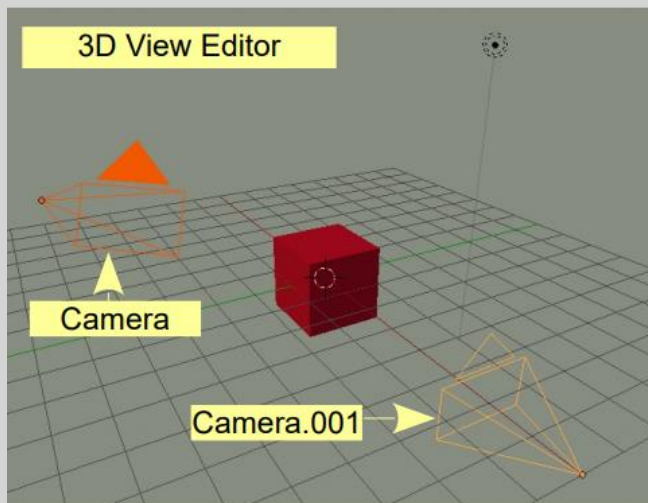


Blender/Python API

Camera Settings

Cameras

If you select Camera and press **Num Pad 0** you will get a Camera View taken by the original Camera. To get Camera View from Camera.001, have Camera.001 selected then press **Ctrl + Num Pad 0**.



Select a Camera press **Ctrl + Num Pad 0**
For Camera View.

Blender/Python API

Camera Settings

Camera Settings

Settings are found in the **Properties Editor, Object Data Properties** buttons for the selected Camera

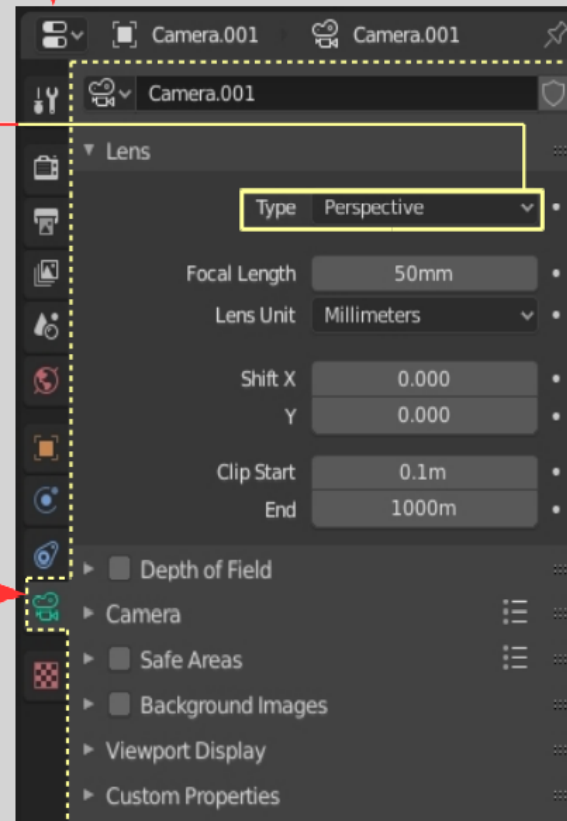
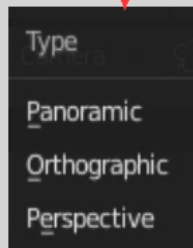
Lens Tab:

Perspective, Orthographic or Panoramic: Used to change the camera from showing a true-life perspective view to an orthographic view.

Focal Length: Sets up a lens length much like a real camera; 35mm is a good safe setting but wide and tight angle settings work for different needs.

Shift: Pushes the camera's view in a direction, without changing perspective.

Properties Editor



Object Data

Blender/Python API

Camera Settings

Tilt-shift lenses

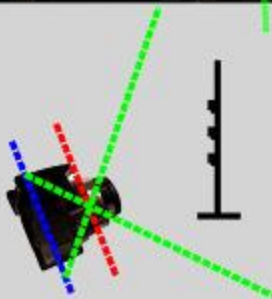
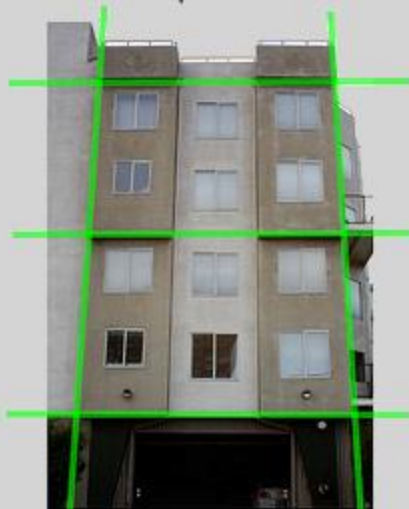




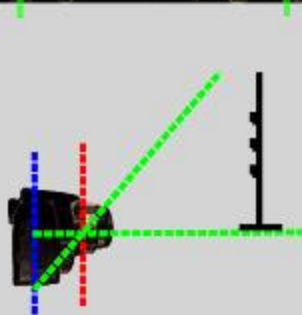
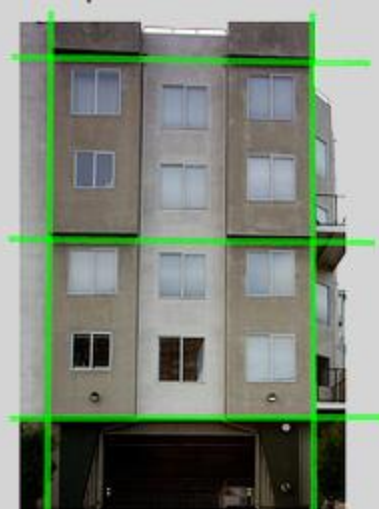
Blender/Python API Camera Settings

Tilt-shift lenses are often used in architectural photography. Parallel lines that appear to converge due to perspective distortion can be realigned to appear parallel.

Uncorrected Perspective



Shifted Perspective



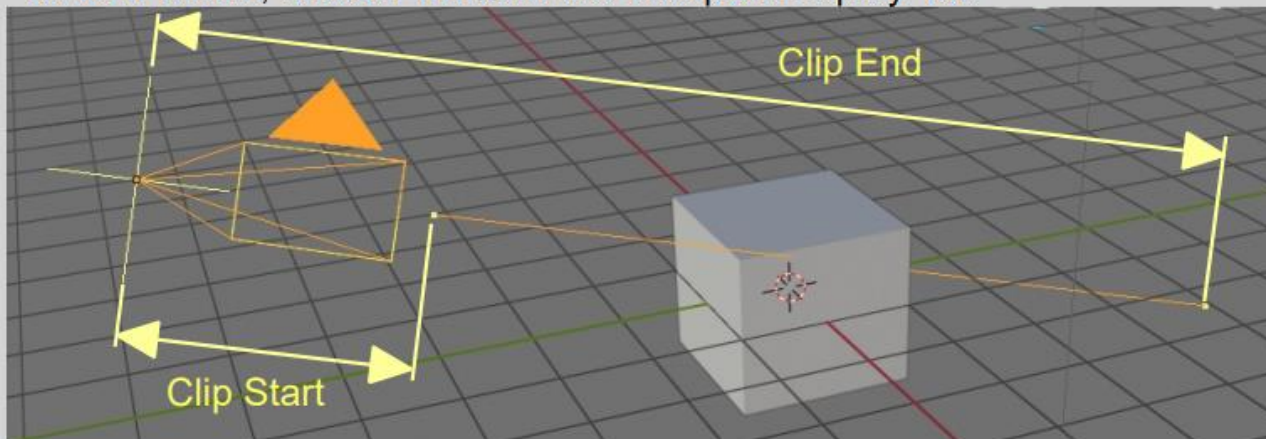


Blender/Python API Camera Settings

Clip Start: How close an object can be to the camera and still be seen

Clip End: How far away objects can be seen by the Camera; in very large Scenes, this needs to be set higher or objects disappear from view

To view Limits, check **Limits** in the Viewport Display Tab

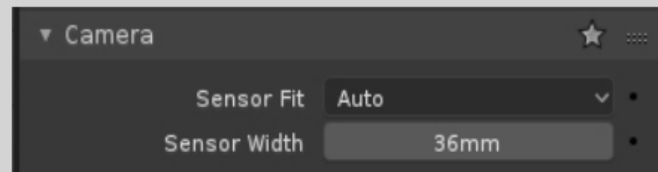




Blender/Python API Camera Settings

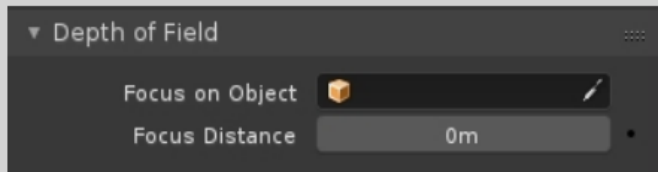
Camera Tab:

Camera Presets: Allows matching the virtual Camera in the Blender Scene with a real camera used to record video. This produces a more realistic effect when Camera Tracking



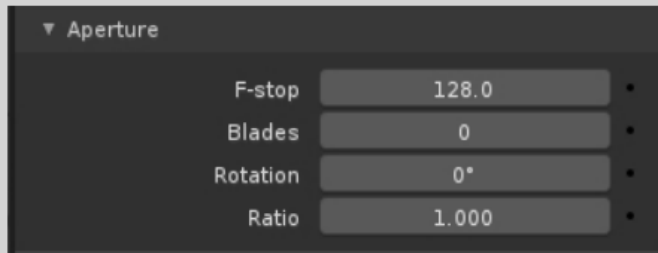
Depth of Field Tab:

Used with **Nodes** to blur foreground and background objects



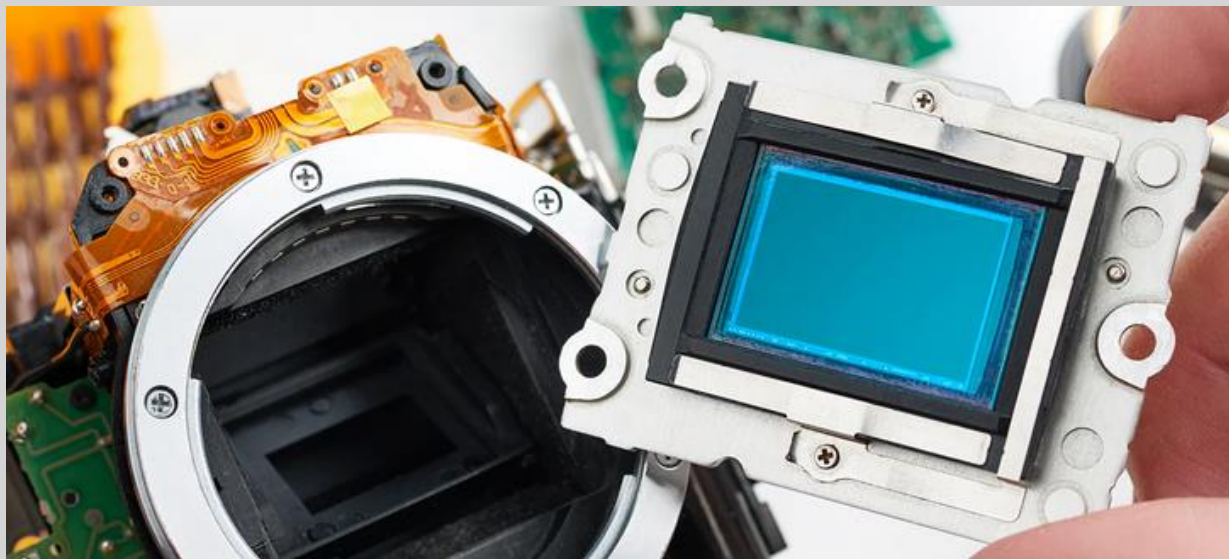
Aperture Tab:

Mimics f-stop settings on a real camera which control the amount of light entering the camera.

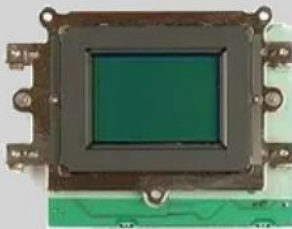


Blender/Python API

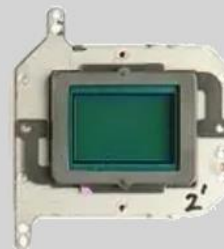
Camera Settings



35mm
24 x 36 mm

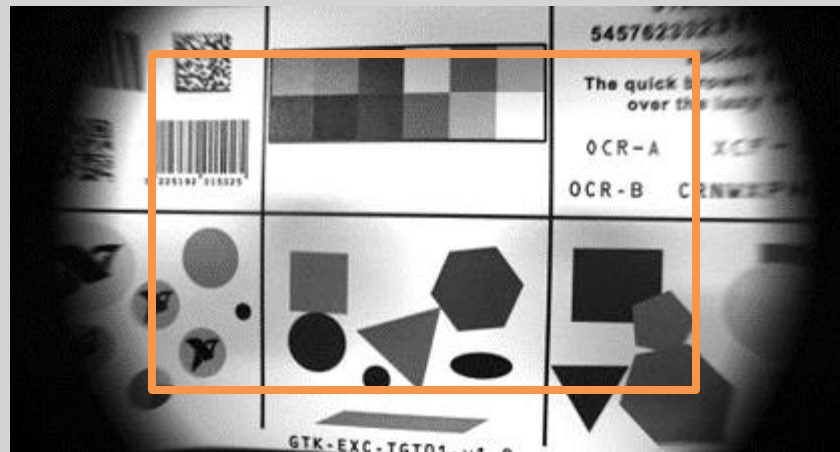
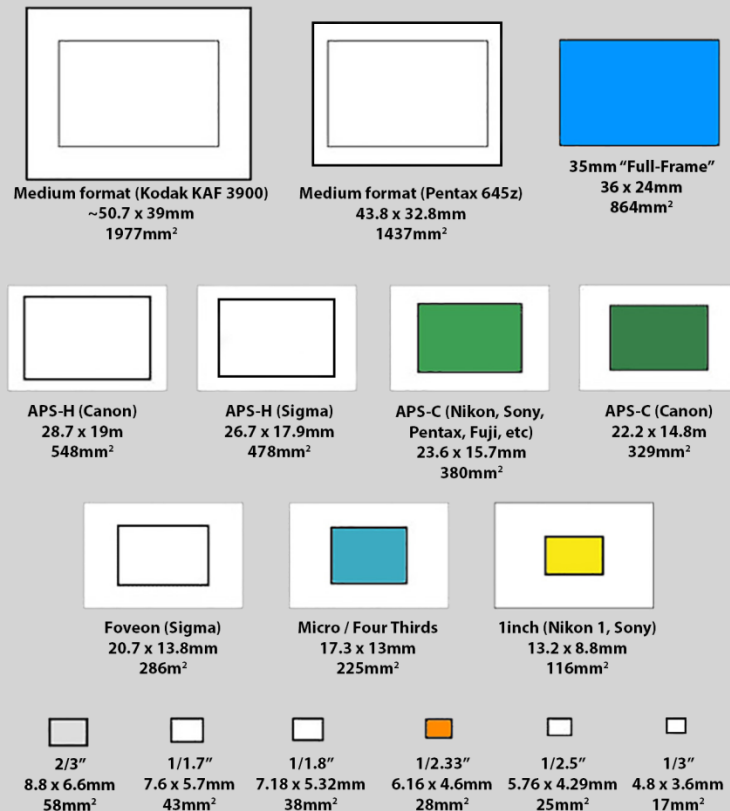


Canon EOS 1D
19 x 29 mm



APS-C
15 x 22 mm

Blender/Python API Camera Settings





Blender/Python API Camera Settings

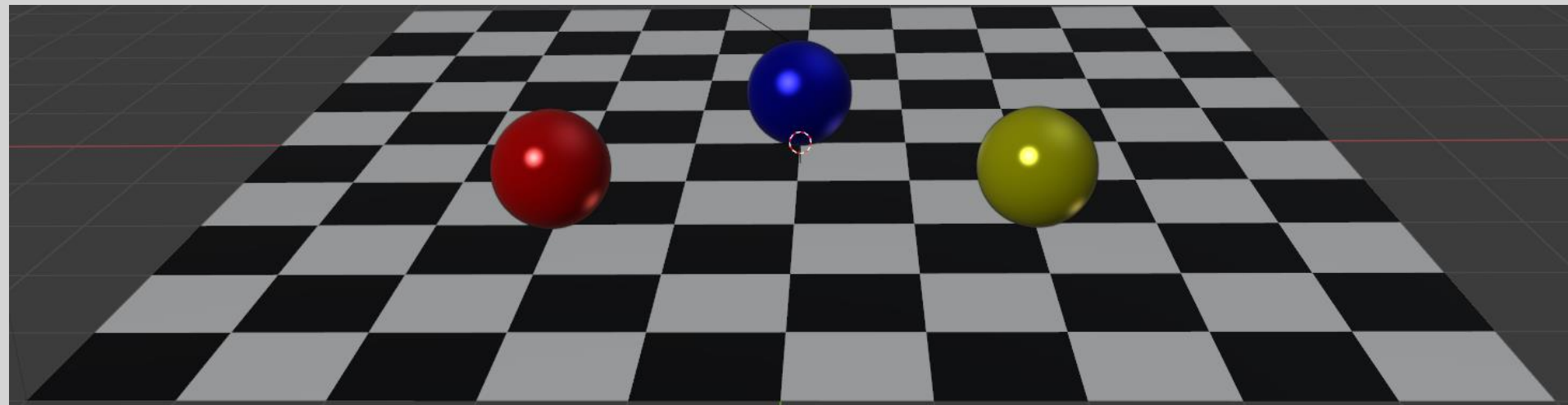
```
def camera(origin, lens=35, clip_start=0.1, clip_end=200, type='PERSP', ortho_scale=6):  
    # Create object and camera  
    bpy.ops.object.camera_add(location=origin)  
    obj = bpy.context.object  
    obj.data.lens = lens  
    obj.data.clip_start = clip_start  
    obj.data.clip_end = clip_end  
    # 'PERSP', 'ORTHO', 'PANO'  
    obj.data.type = type  
    if type == 'ORTHO':  
        obj.data.ortho_scale = ortho_scale  
    return obj
```



Blender/Python API

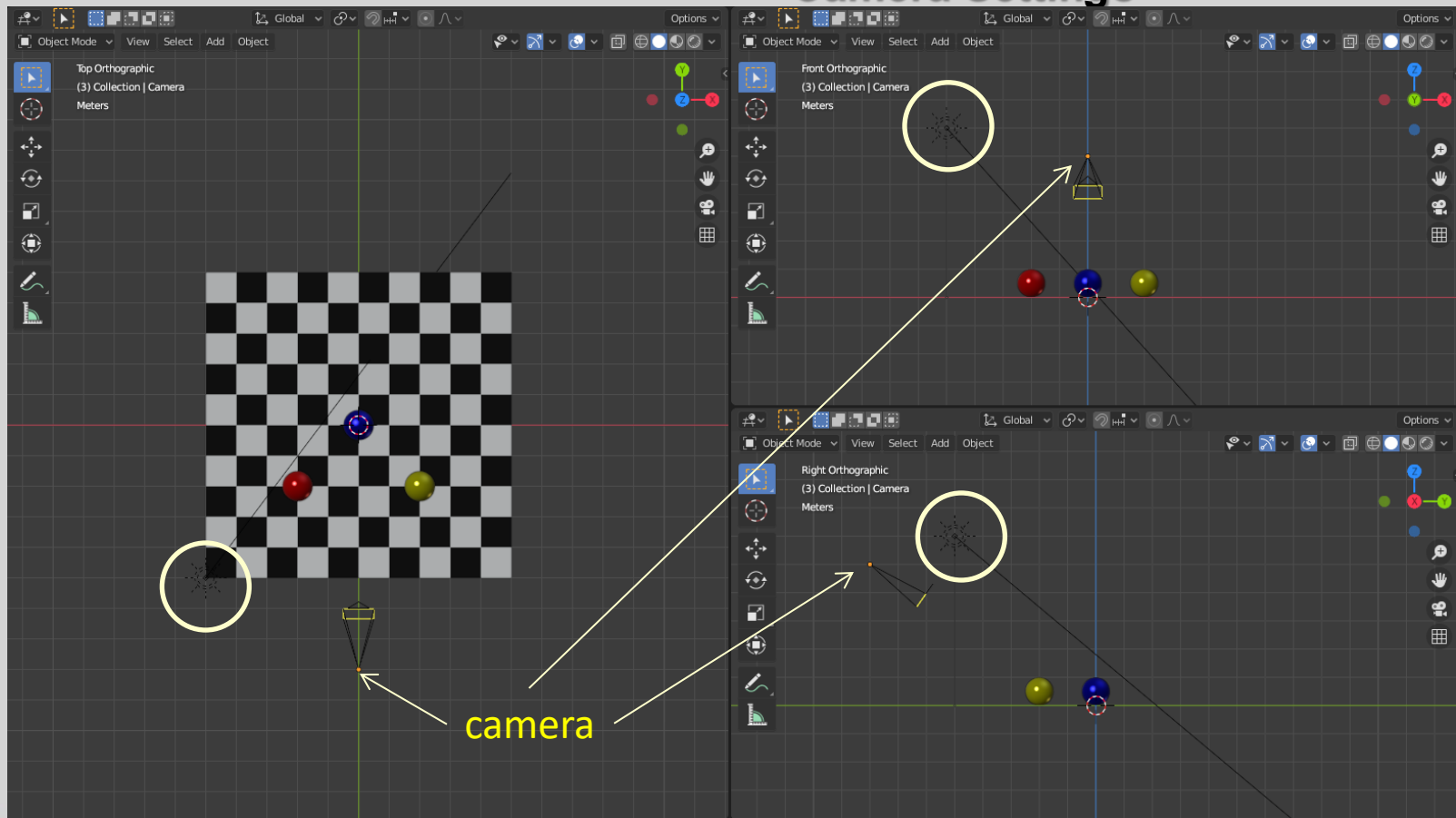
Camera Settings

Create a 10x10 (width and depth) checkerboard plane and add three spheres with different color.





Blender/Python API Camera Settings





Blender/Python API Camera Settings

```
tk.create.sphere('RedSphere')  
tk.sel.scale((0.5,)*3)  
bpy.context.object.data.materials.append(red)  
tk.sel.translate((-2, -2, 0.5))  
tk.setSmooth(bpy.context.object, level = 1, smooth = True)
```

```
tk.create.sphere('BlueSphere')  
tk.sel.scale((0.5,)*3)  
bpy.context.object.data.materials.append(blue)  
tk.sel.translate((0, 0, 0.5))  
tk.setSmooth(bpy.context.object, level = 1, smooth = True)
```

```
tk.create.sphere('YellowSphere')  
tk.sel.scale((0.5,)*3)  
bpy.context.object.data.materials.append(yellow)  
tk.sel.translate((2, -2, 0.5))  
tk.setSmooth(bpy.context.object, level = 1, smooth = True)
```



Blender/Python API Camera Settings

```
whiteLamp = tk.create.light((-5, -5, 6), 'SUN', 10, (1, 1, 1))  
tk.sel.rotate_x(radians(30))  
tk.sel.rotate_y(radians(50))  
tk.sel.rotate_z(radians(-90))
```



```
camera35mm = tk.create.camera((0, -8, 5), lens=35, clip_start=0.1,  
clip_end=200, type='PERSP', ortho_scale=6)  
tk.sel.rotate_x(radians(55))
```

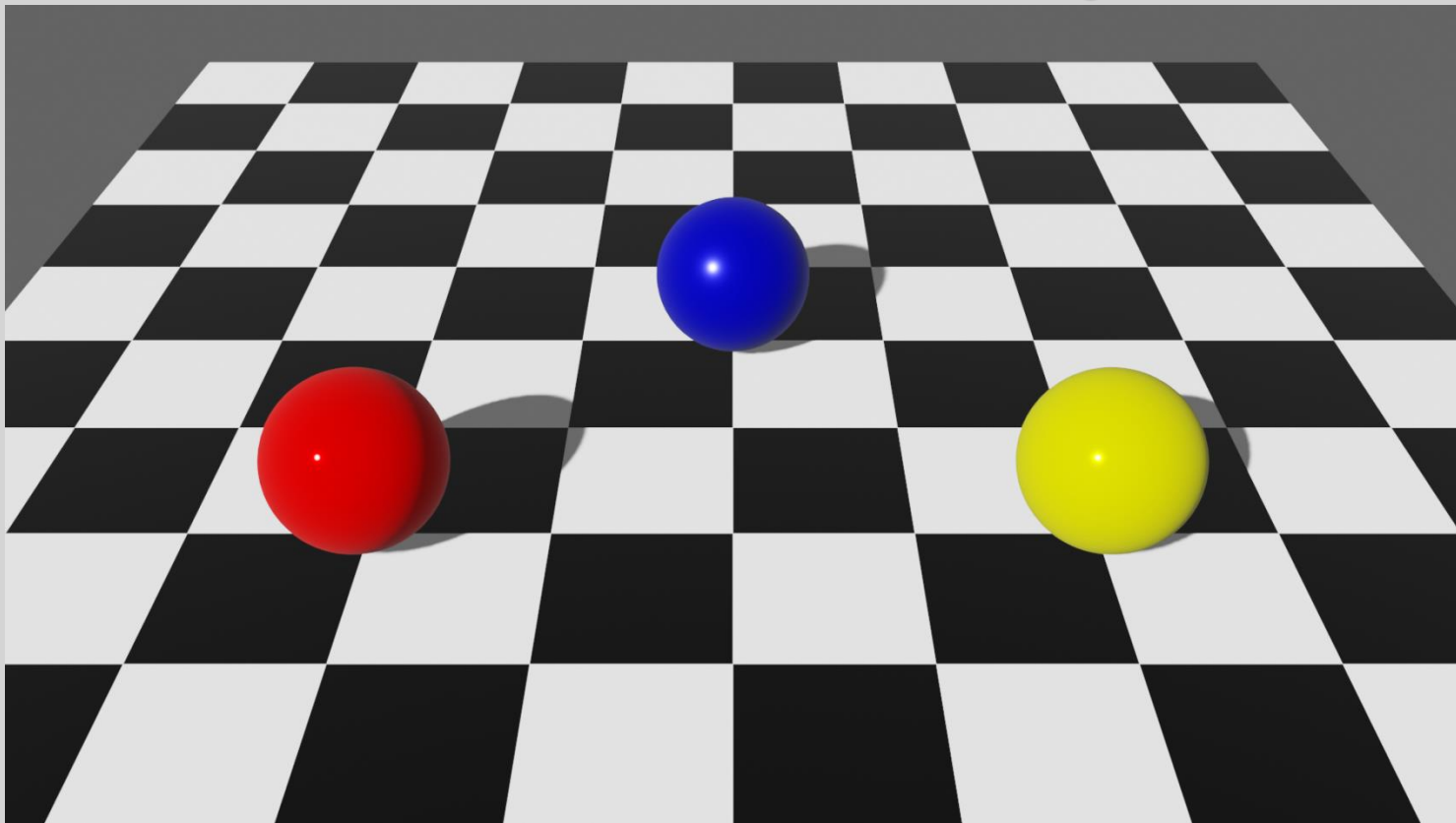
```
# light gray background
```

```
bpy.data.worlds["World"].node_tree.nodes["Background"].inputs[0].default_value  
= (0.1, 0.1, 0.1, 1)
```



Blender/Python API

Camera Settings

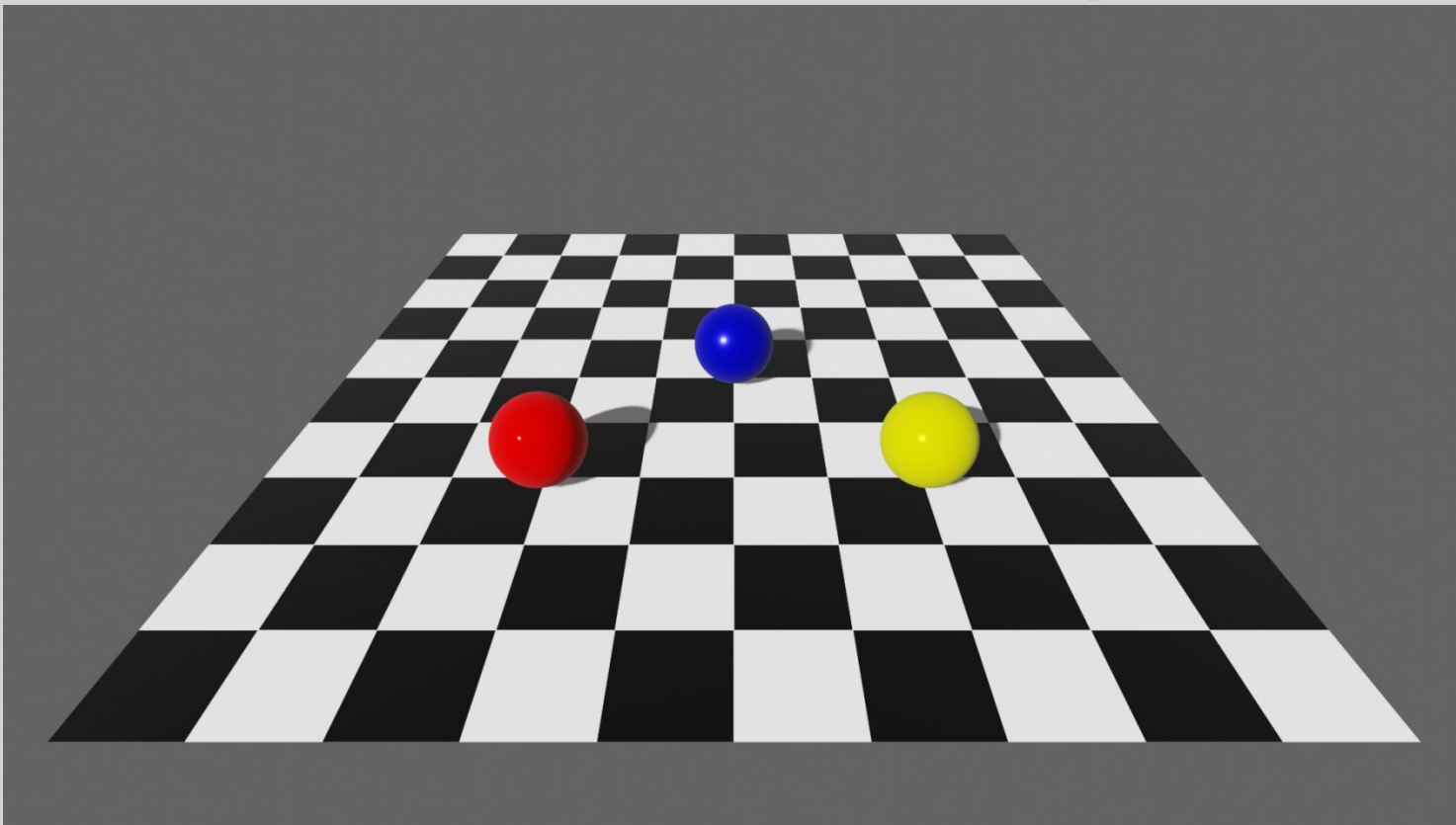


35 mm



Blender/Python API

Camera Settings

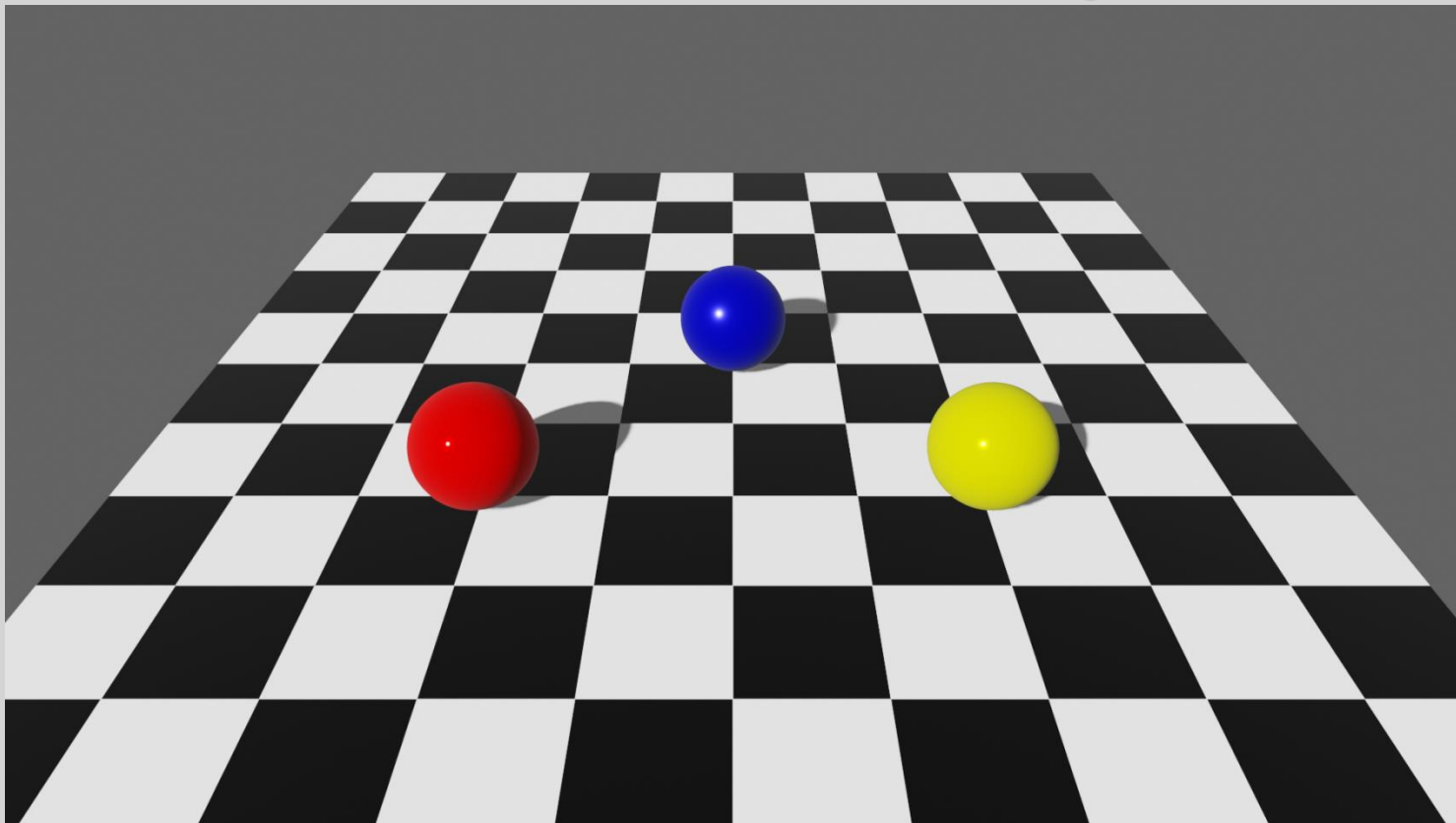


18 mm



Blender/Python API

Camera Settings

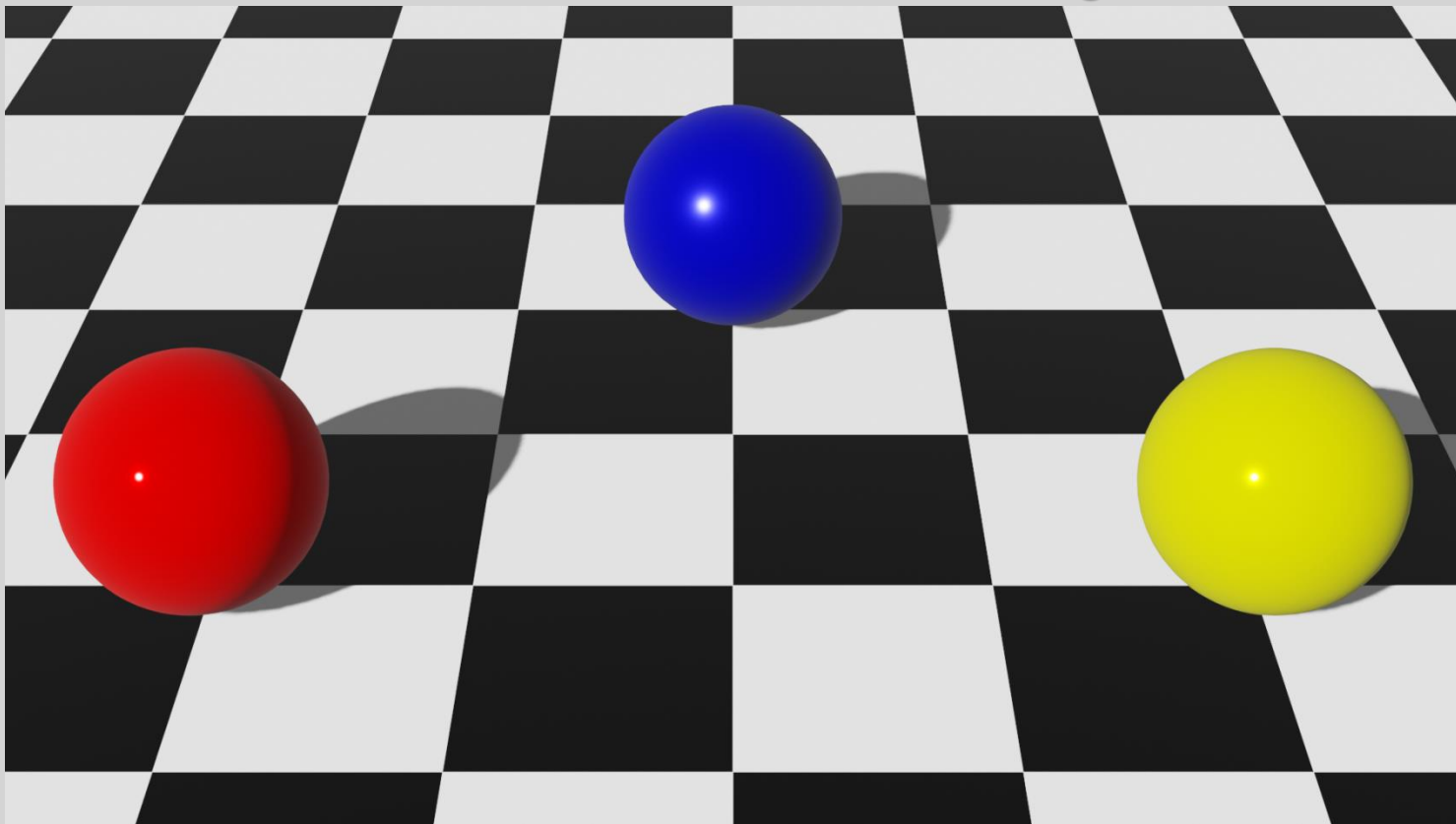


24 mm



Blender/Python API

Camera Settings

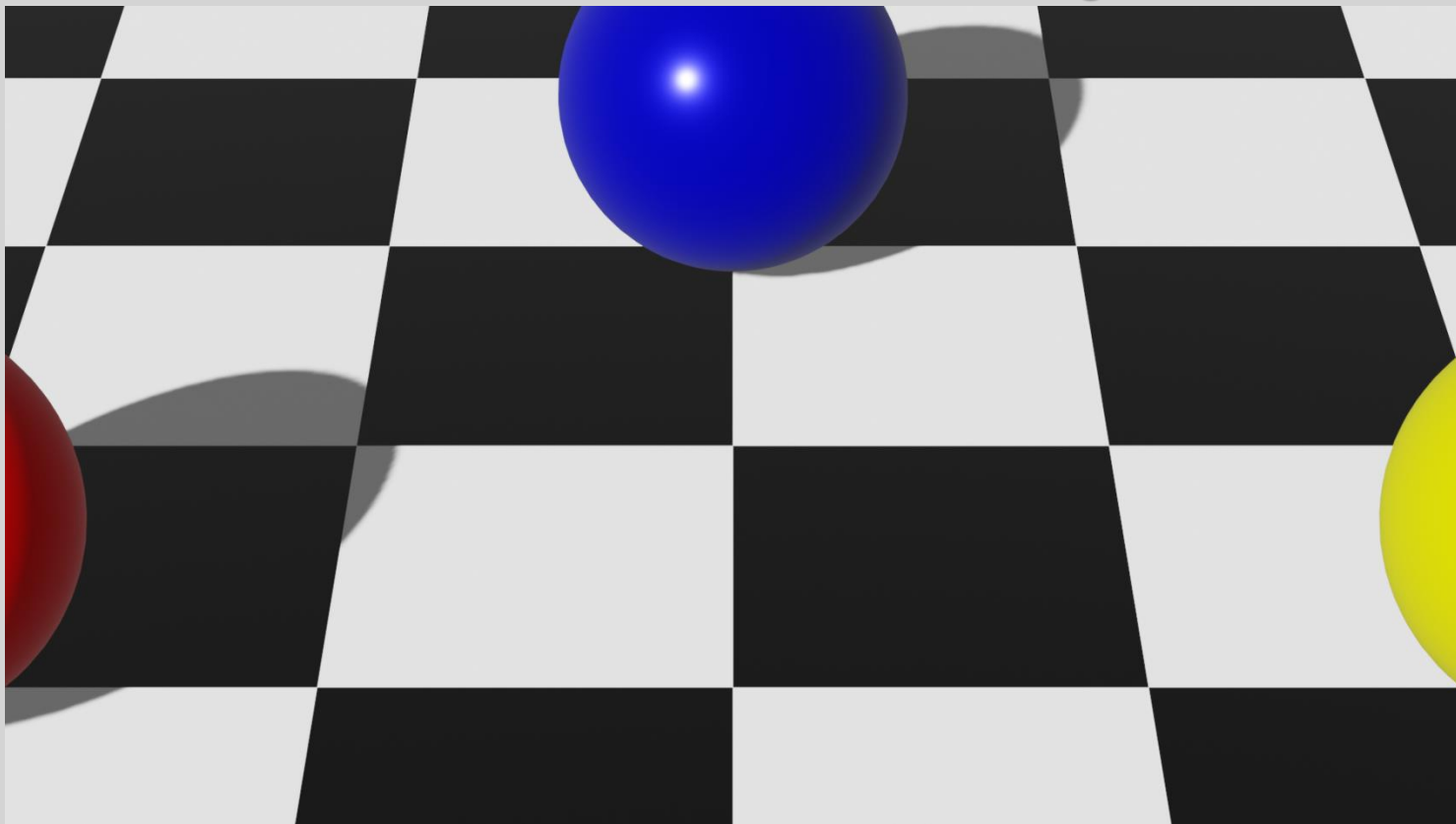


50 mm



Blender/Python API

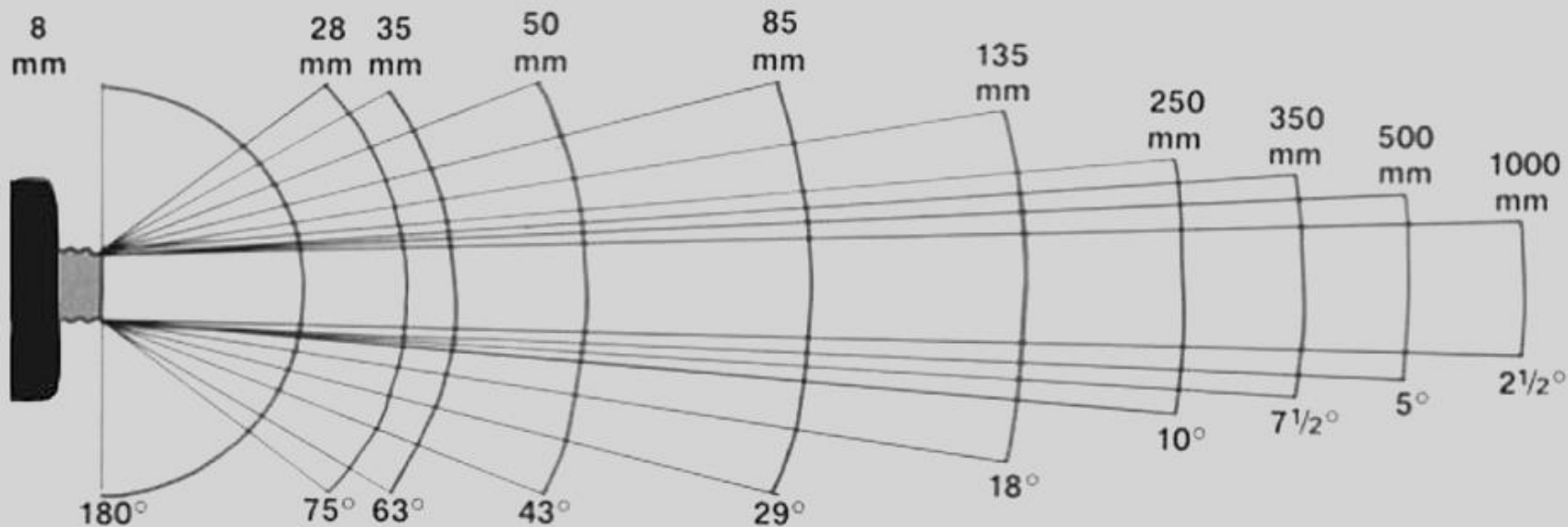
Camera Settings



80 mm

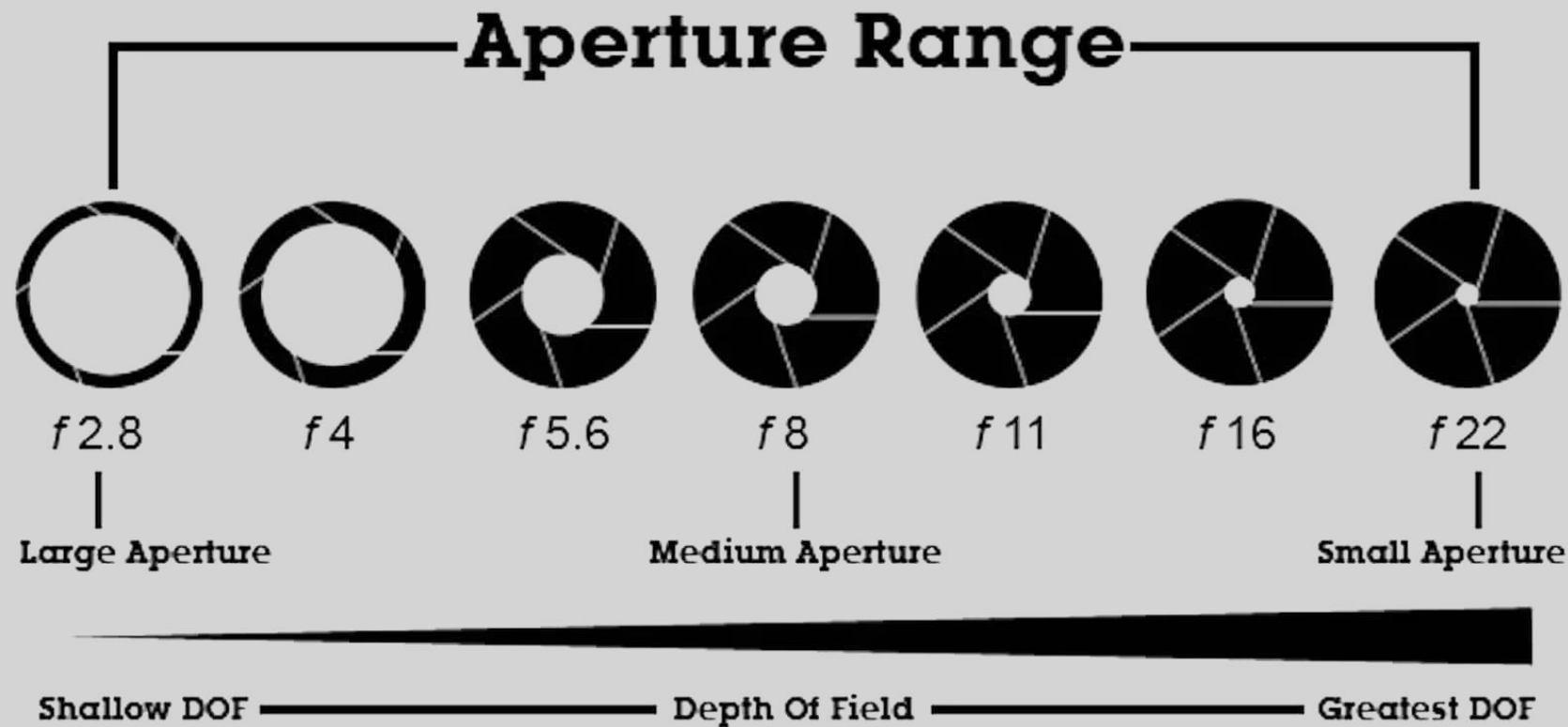


Blender/Python API Camera Settings



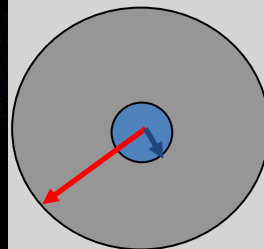
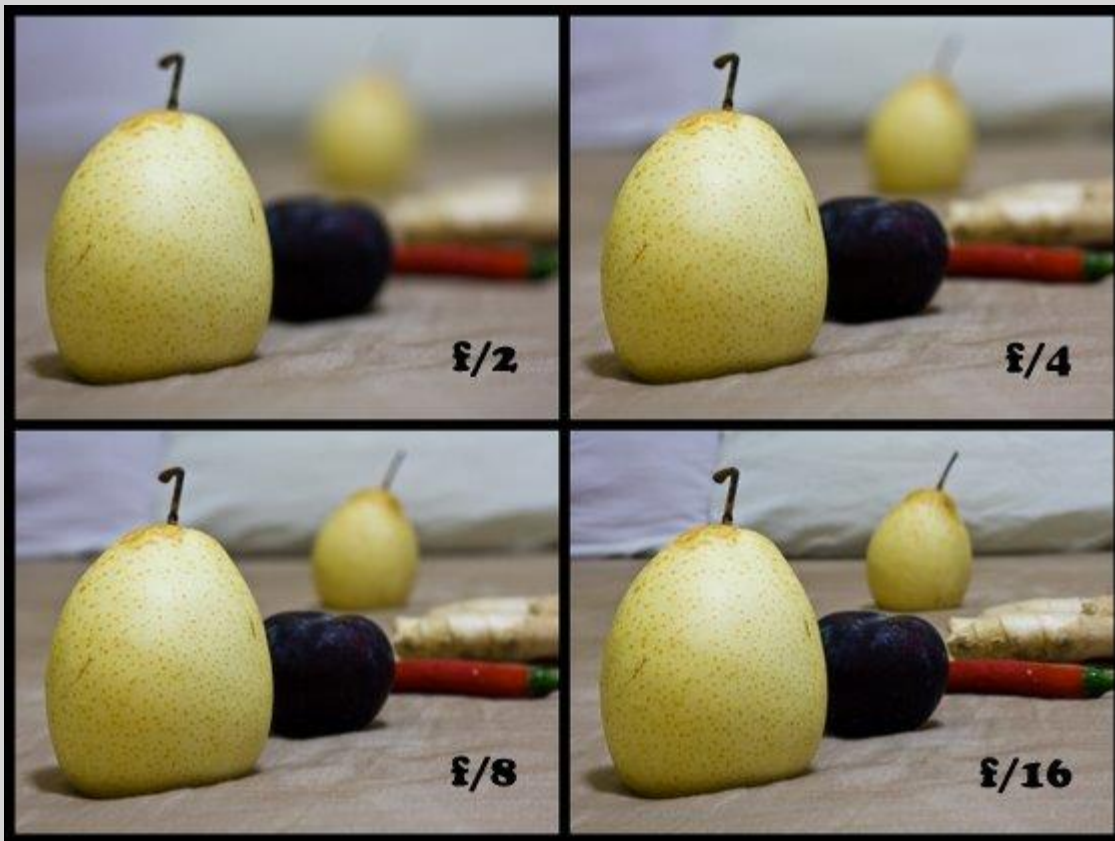
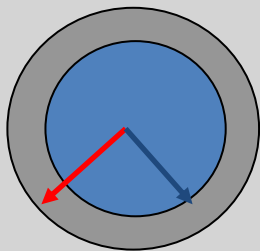


Blender/Python API Camera Settings






Blender/Python API Camera Settings



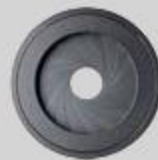


Blender/Python API Camera Settings

```
myCamera = tk.create.camera((0, -5, 0.5), lens=50, clip_start=0.1,  
clip_end=200, type='PERSP', ortho_scale=6)  
myCamera.data.dof.use_dof = True  
myCamera.data.dof.aperture_fstop =  f/2  
myCamera.data.dof.focus_distance = 3.0
```



f/2



f/8



f/2.8



f/11



f/4



f/16



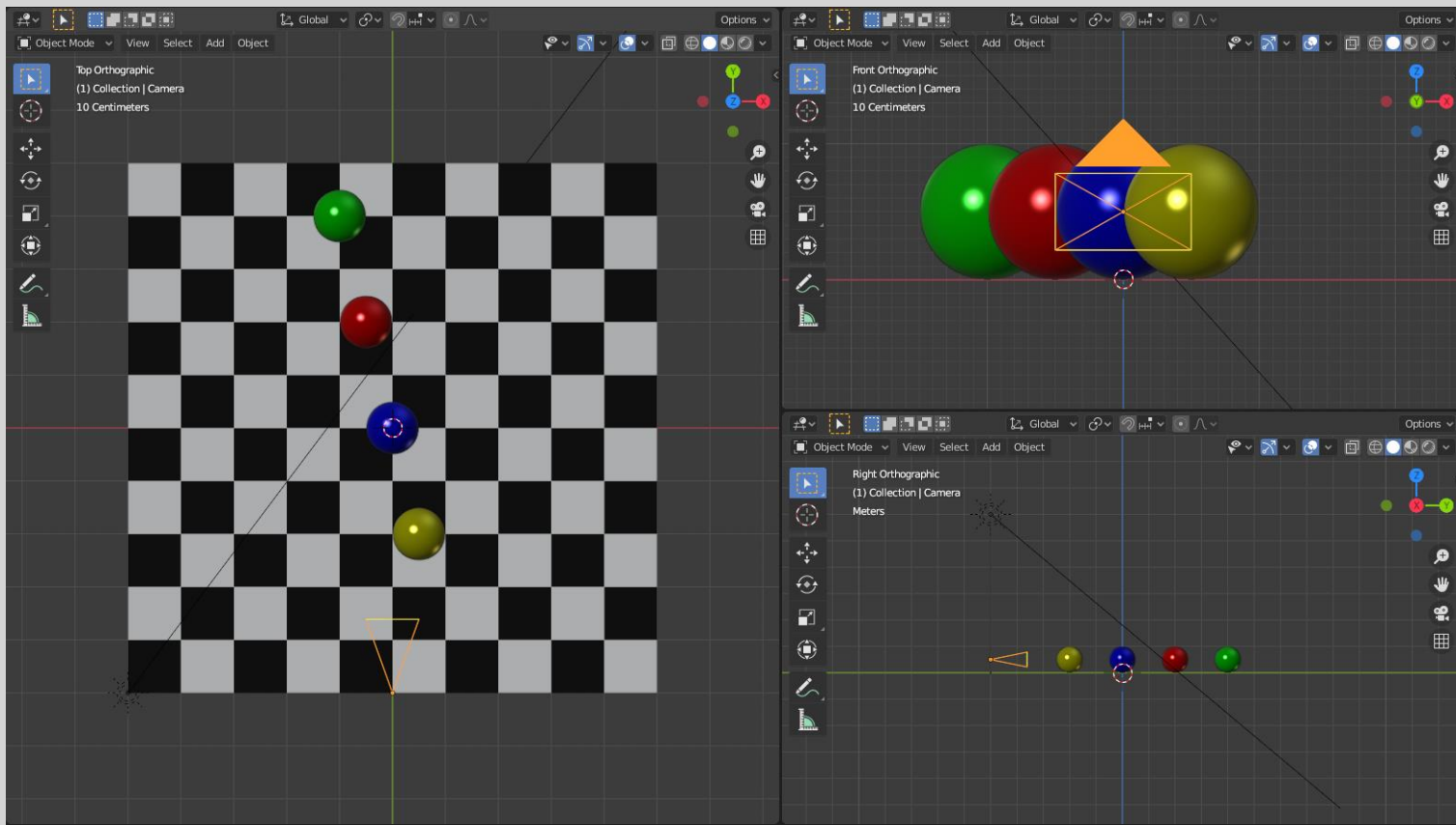
f/5.6



f/22



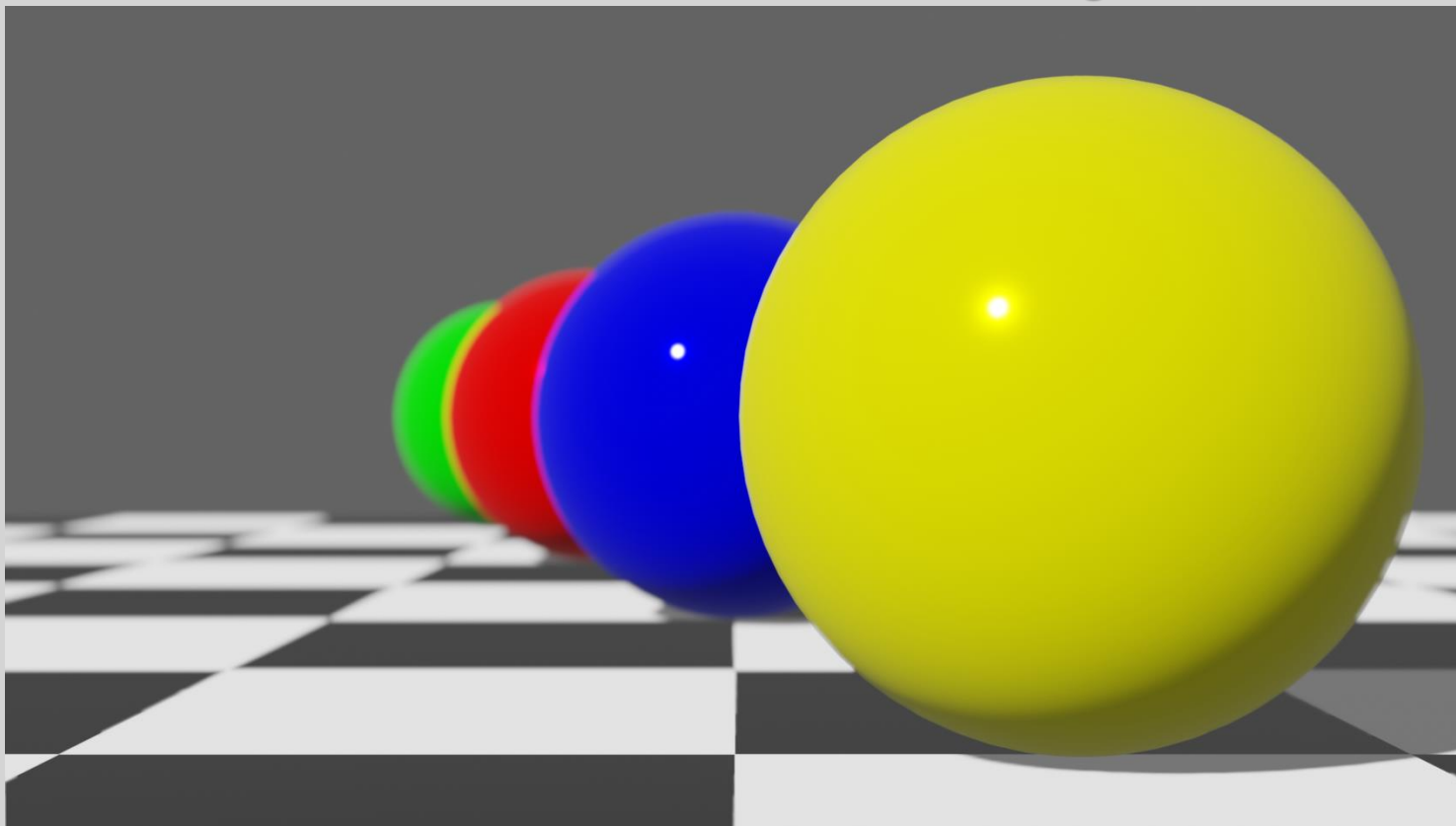
Blender/Python API Camera Settings





Blender/Python API

Camera Settings



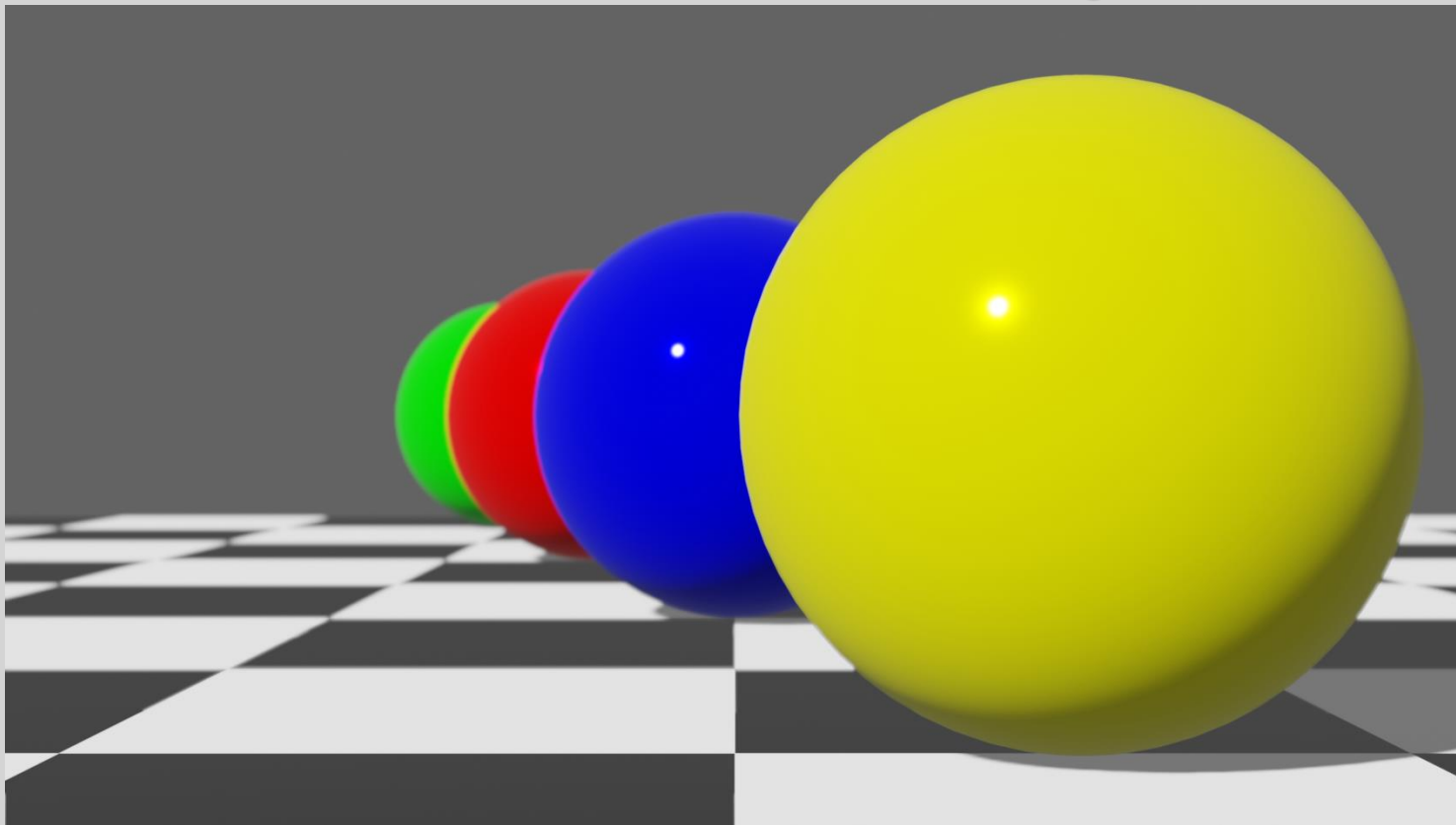
Focus 3m

f 1.4



Blender/Python API

Camera Settings



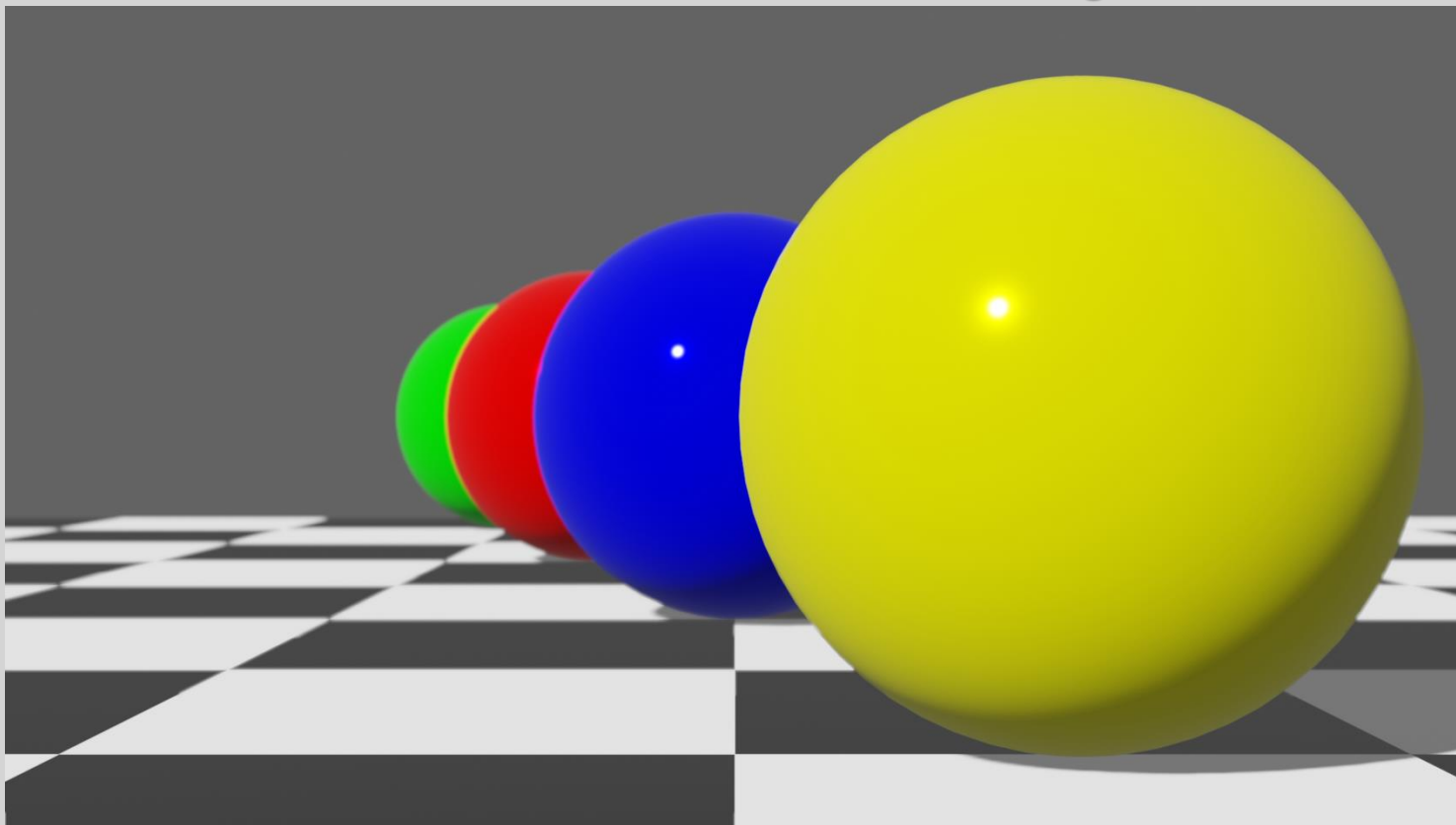
Focus 3m

f 2.8



Blender/Python API

Camera Settings



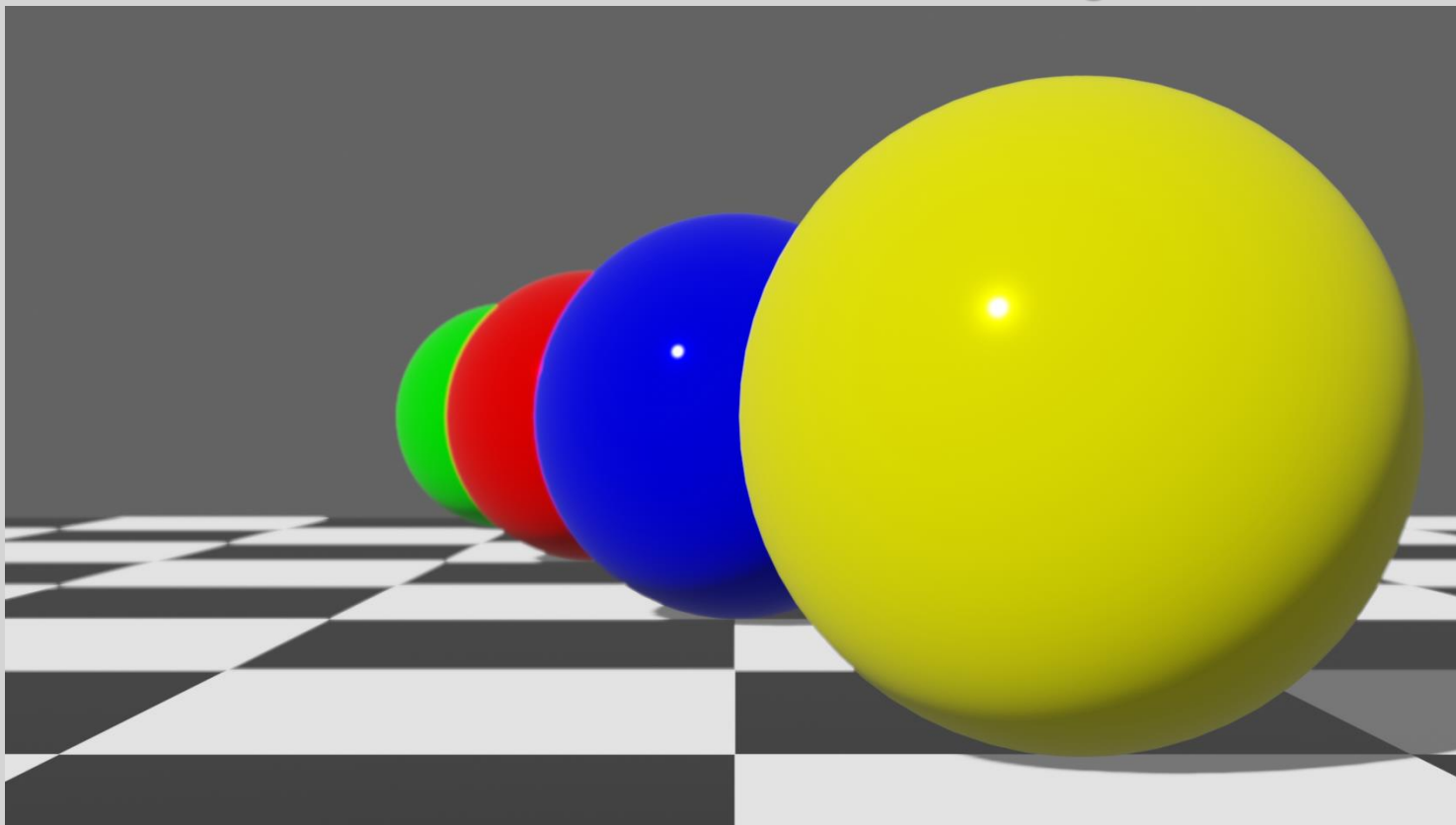
Focus 3m

f 4



Blender/Python API

Camera Settings



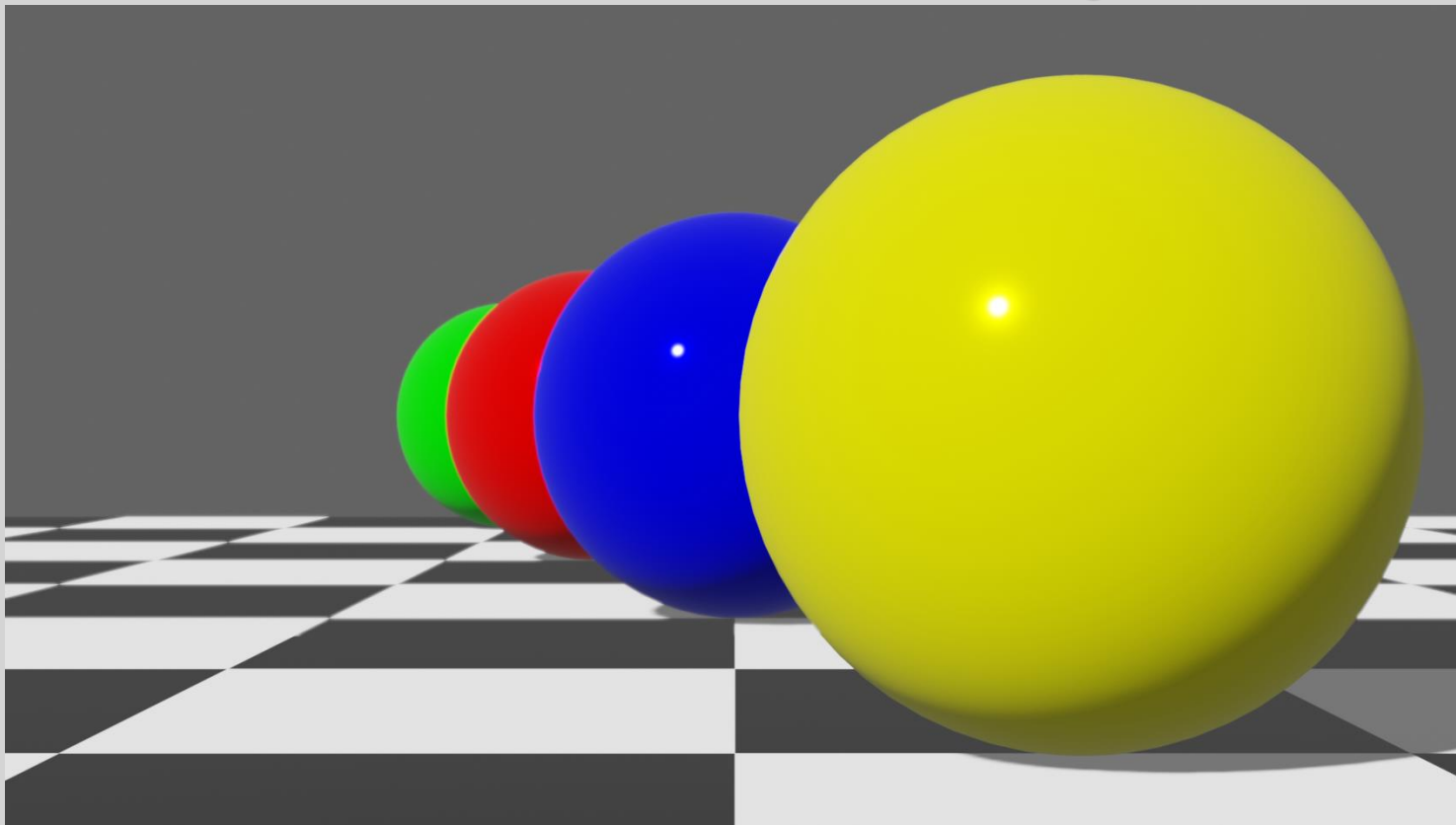
Focus 3m

f 5.6



Blender/Python API

Camera Settings



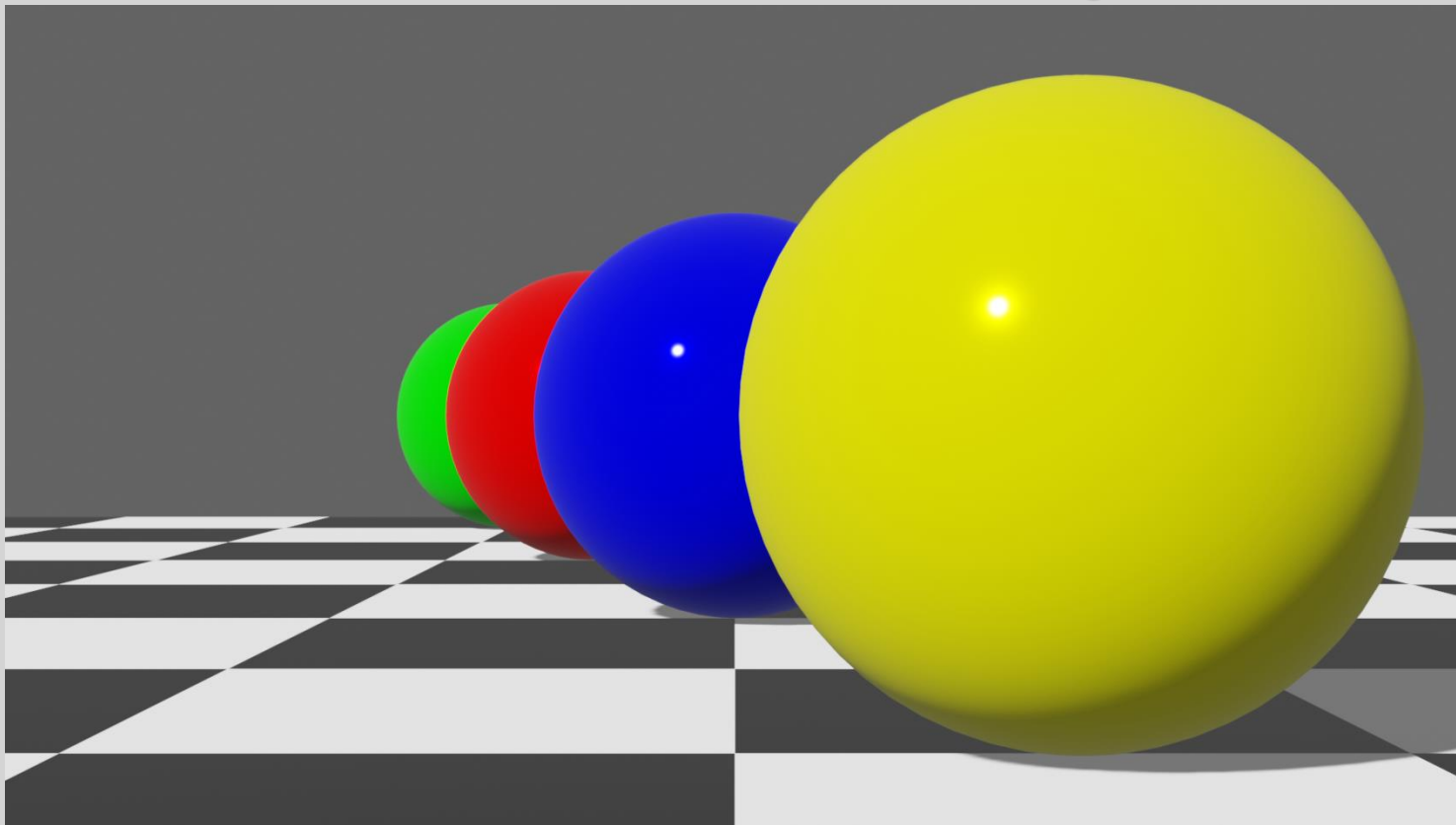
Focus 3m

f 8



Blender/Python API

Camera Settings



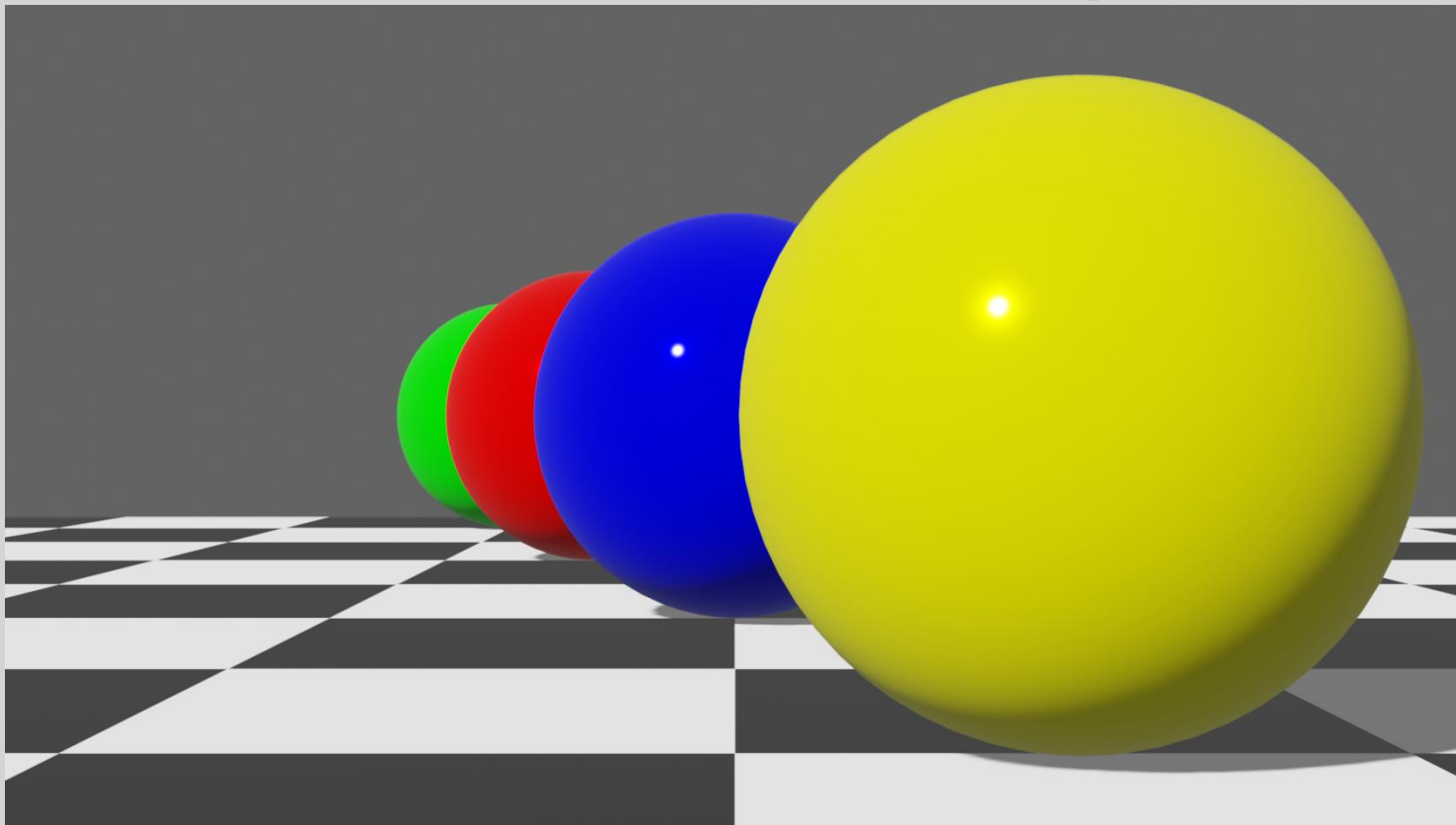
Focus 3m

f 16



Blender/Python API

Camera Settings



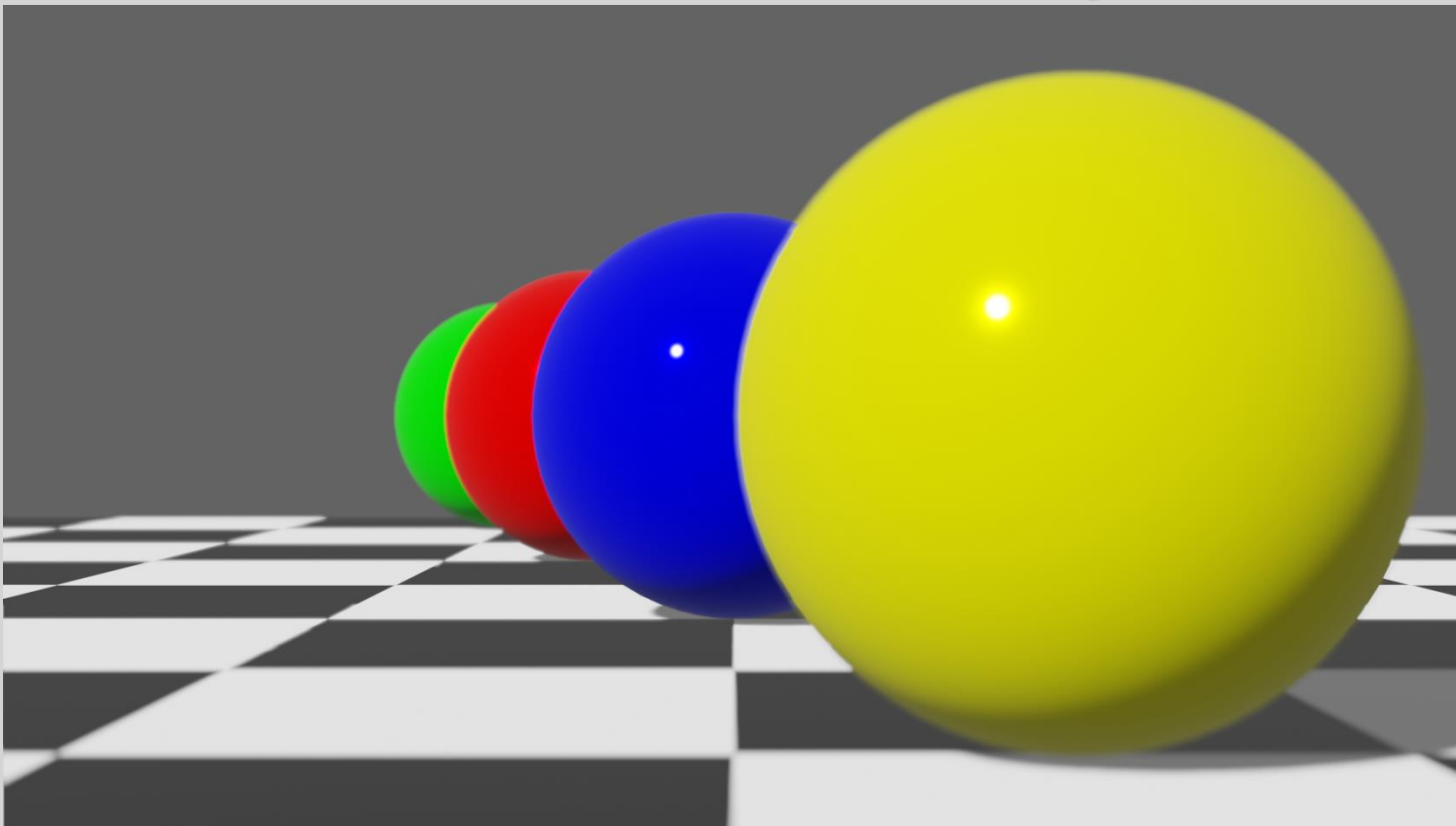
Focus 3m

f 22



Blender/Python API

Camera Settings



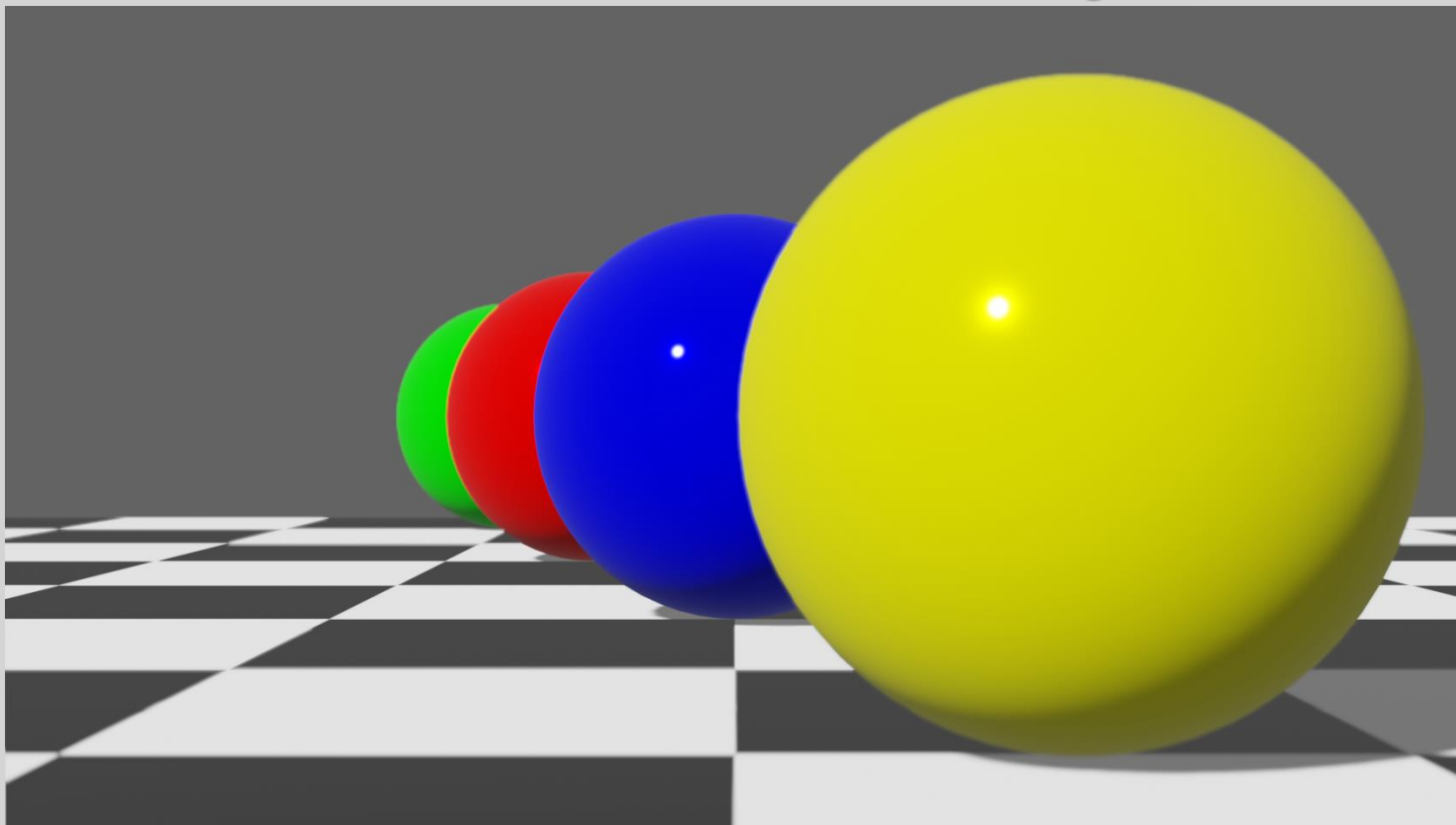
Focus 6m

f 1.4



Blender/Python API

Camera Settings



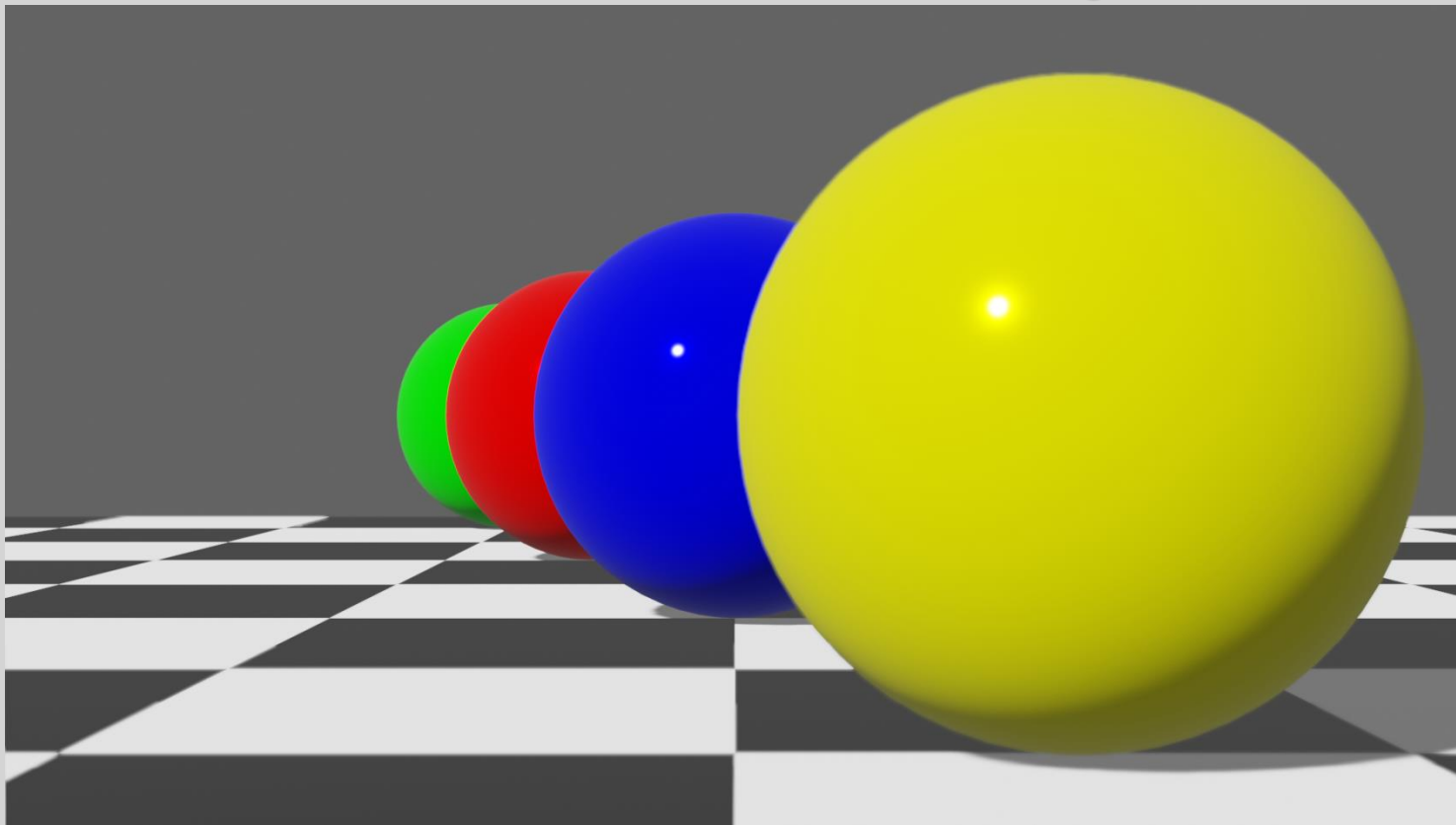
Focus 6m

f 2.8



Blender/Python API

Camera Settings



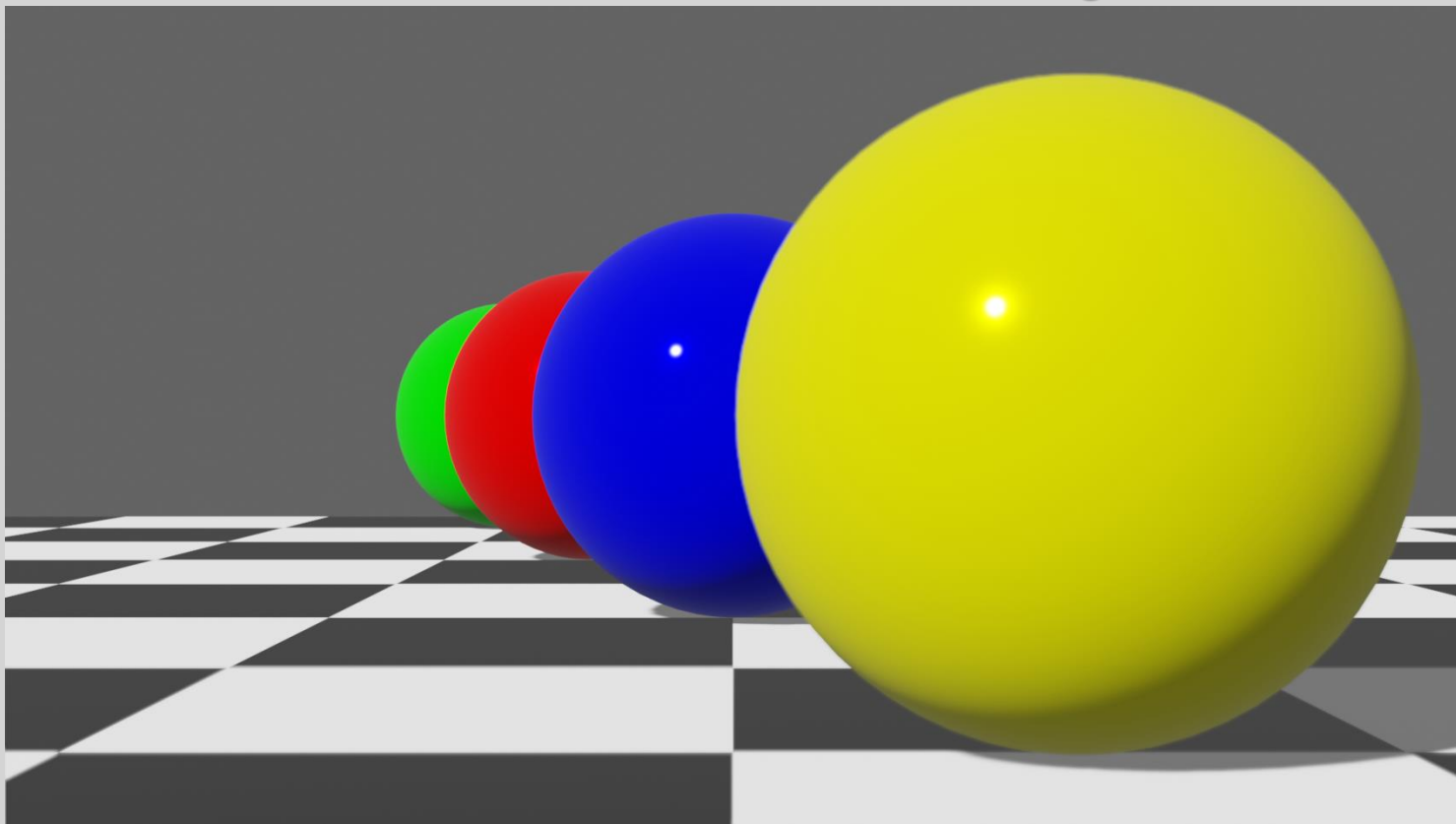
Focus 6m

f 4



Blender/Python API

Camera Settings



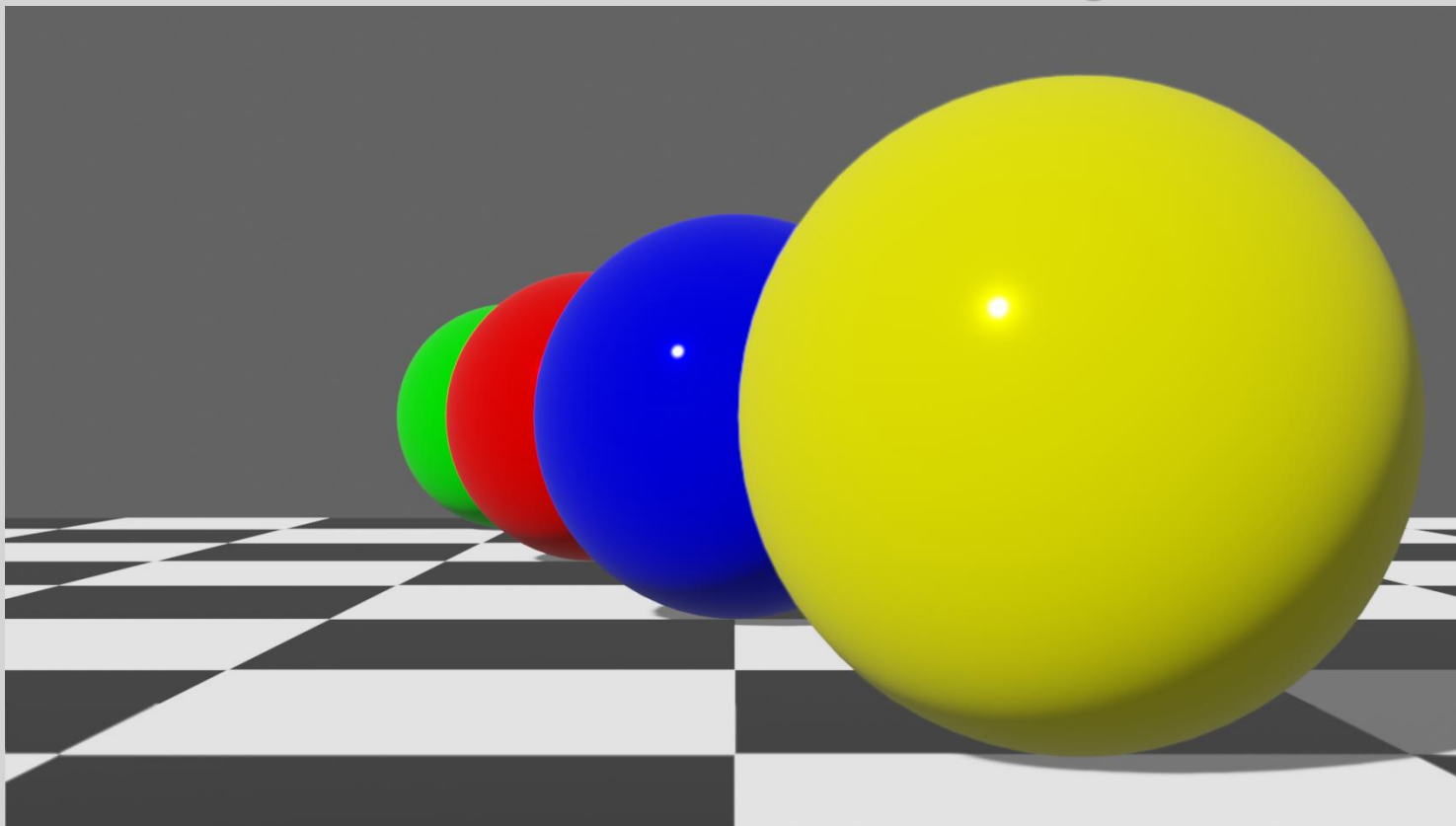
Focus 6m

f 5.6



Blender/Python API

Camera Settings



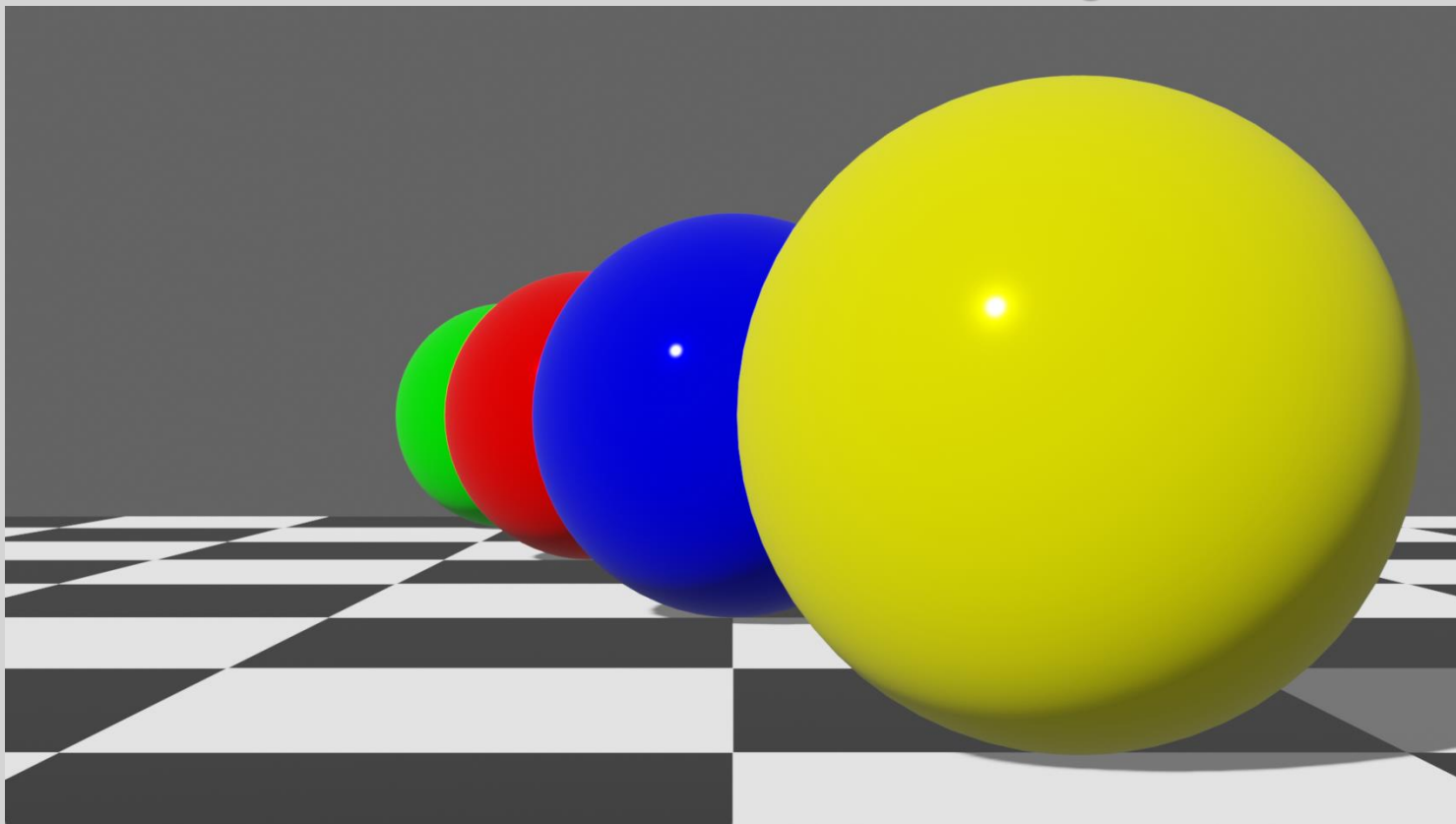
Focus 6m

f 8



Blender/Python API

Camera Settings



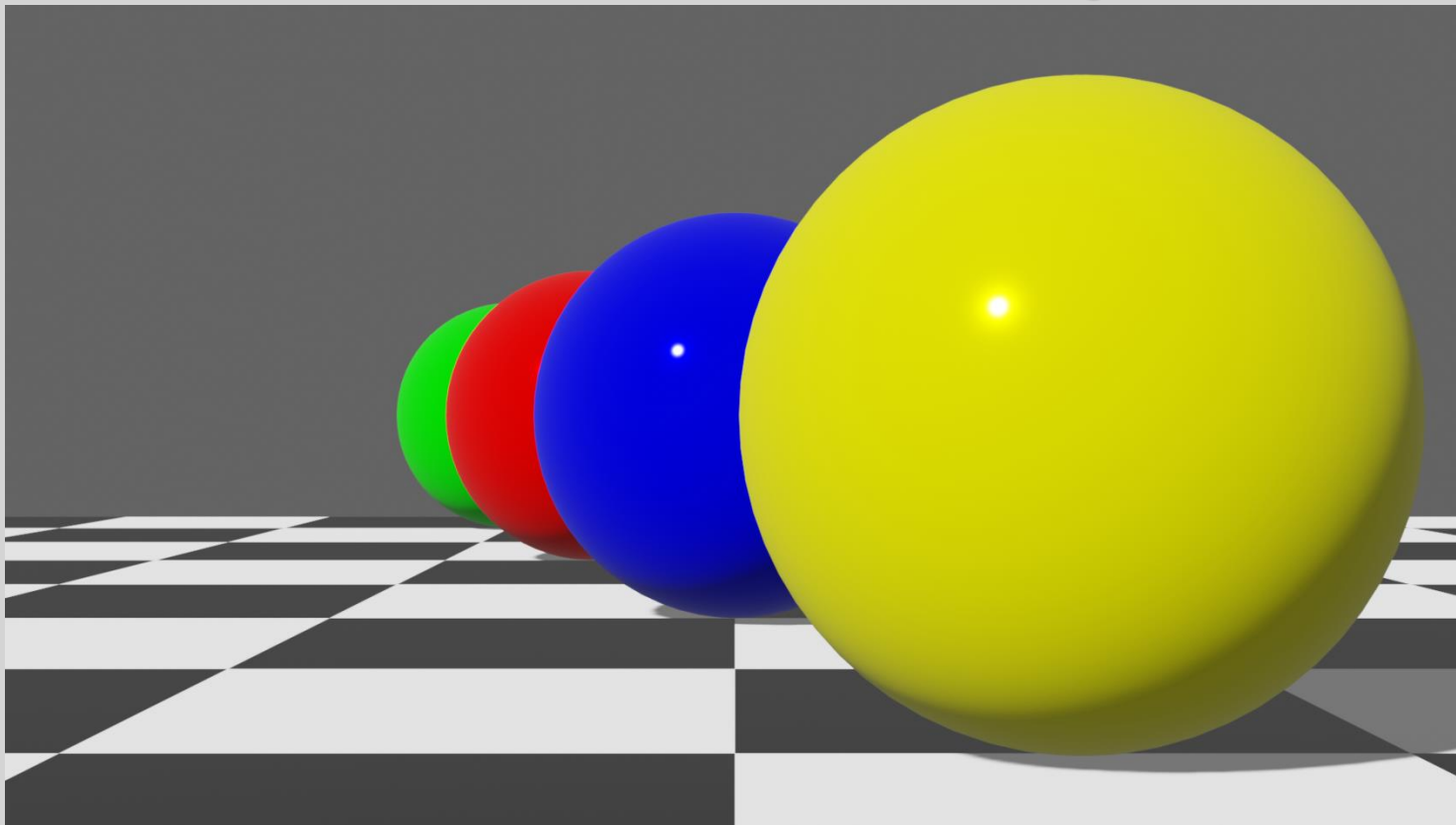
Focus 6m

f 16



Blender/Python API

Camera Settings



Focus 6m

f 22



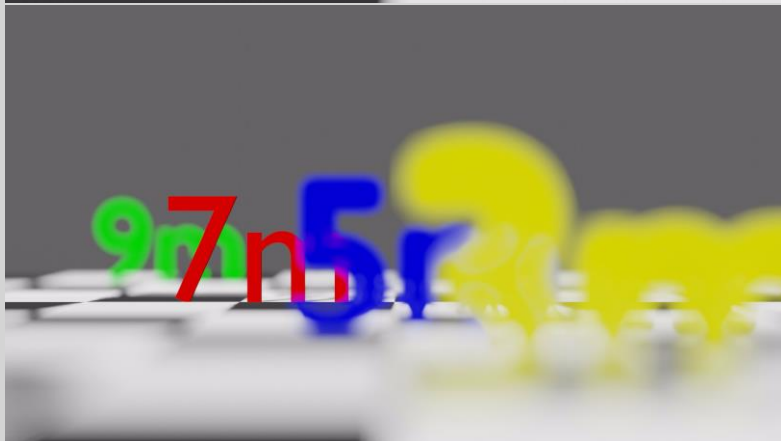
Blender/Python API

Camera Settings





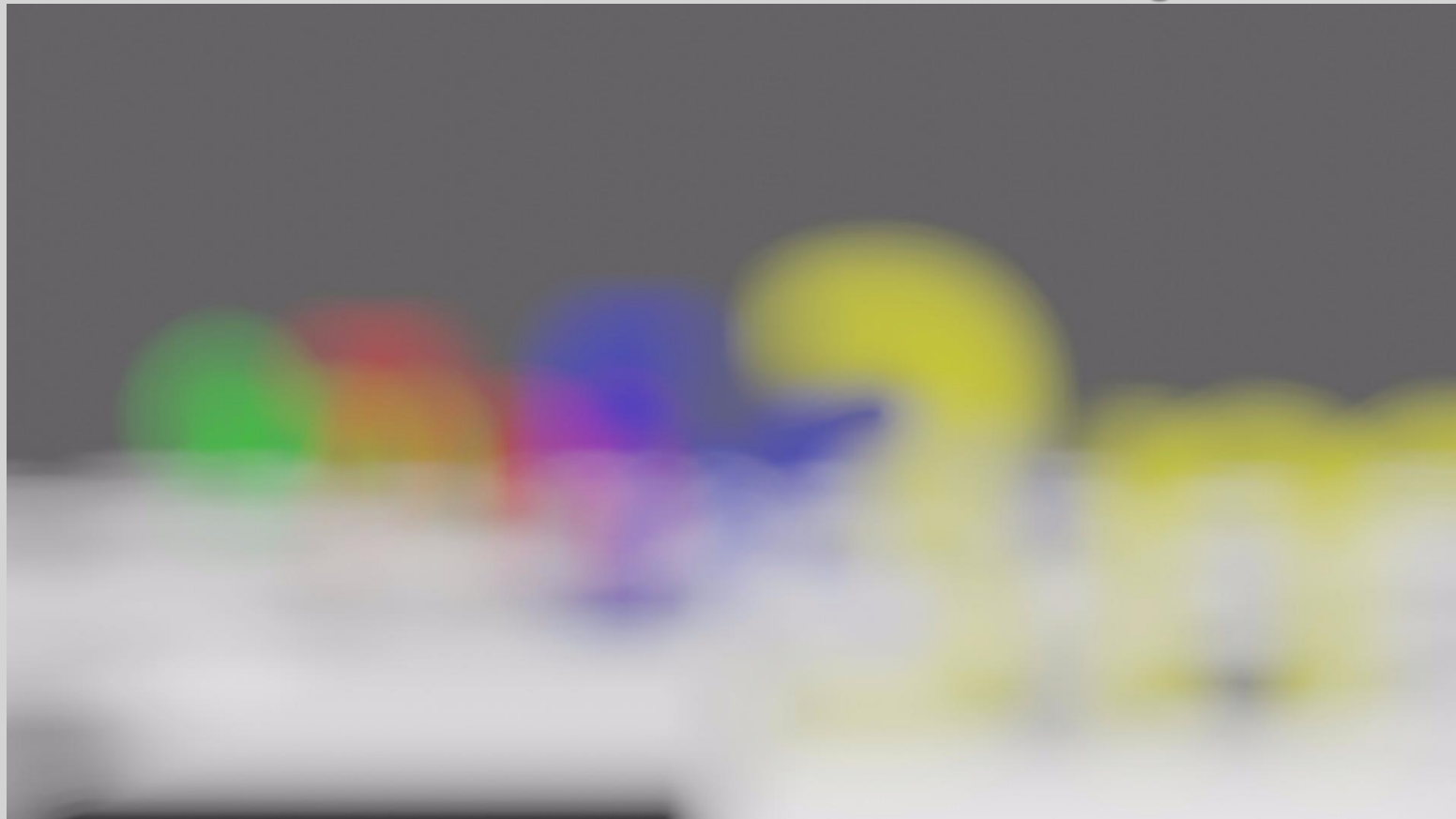
Blender/Python API Camera Settings





Blender/Python API

Camera Settings





Blender/Python API Render Function

Rendering is the process of converting the Blender file information into an Image file or a Video Movie file. In practice this entails taking the data producing what you see in Camera View in Blender and converting it into image or Video file format. There are three separate rendering systems in Blender. One is the Blender **Eevee** system, the second is the **Cycles** system and the third is **Workbench**.

This lecture explains the basic render procedure using the default **Eevee** system. **Eevee** is a fully-featured PBR (physically based-rendering) engine for real-time visualization. With advanced features such as volumetrics, screen-space reflections and refractions, subsurface scattering, soft and contact shadows, post-processing effects such as ambient occlusion, depth of field, camera motion blur and bloom.



Blender/Python API

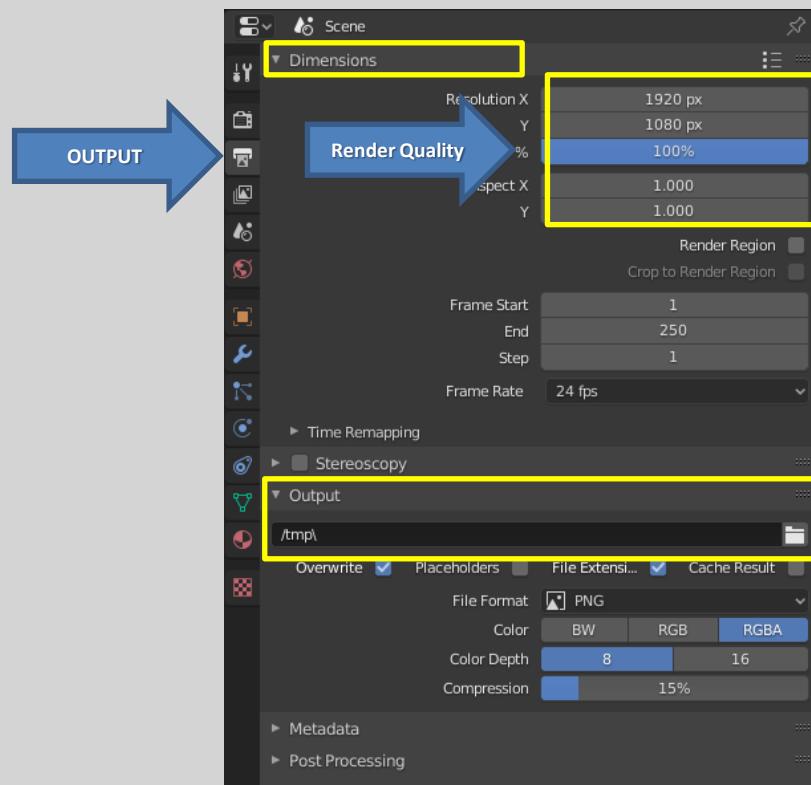
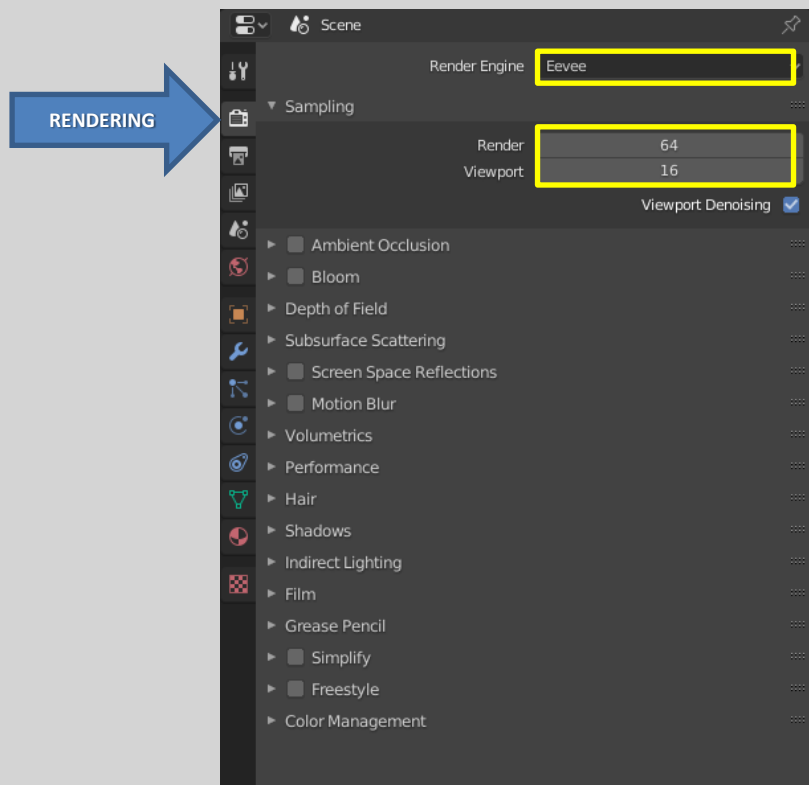
Render Function

Rendering, in the practical sense, converts the data producing what is seen in Camera View in the 3D Viewport Editor into a still image or in the case of an animation into a video clip. To Render a still image press the **F12** button on the Keyboard. An image is Rendered and displayed in a full Screen version of the Image Editor. Press Esc to cancel. To Render an **animation** press **Ctrl + F12**.

Rendering a still image does not save a file to the computer. With the default settings in the Properties Editor, Render buttons, Rendering a still image merely displays the Render in the Image Editor. Rendering an animation, however, saves an image for each Frame in the animation and by default, saves to the `/tmp\` directory as seen in the Render buttons, Output Tab. On a *Windows 10 or 11* Computer this means the files are saved in Local Disk `C:\tmp`.

Blender/Python API

Render Function



The default Resolution 1920 x 1080 equates to the HDTV 1080p **Preset**.



Blender/Python API Render Function

Frame Size	Aspect Ratio	Description (note these are only the most common formats)
1920x1080	16x9	1080p/i
1440x1080	16x9	1080i (Most HDV use this format)
1280x720	16x9	720p
852x480	16x9	480p
720x480	4:3	DV NTSC (when the pixels are square it is actually 3:2)
720x480	16:9*	DV NTSC / Anamorphic* / Wide Screen (non square pixles)
720x576	5:4	DV PAL
640x480	4:3	a ration suitable for square size pixle multimeida video.
640x360	16:9	a ration suitable for square size pixle multimeida thats widescreen.
480x360	4:3	Multimedia large (480x360 : 75%(640x480))
480x270	16:9	Multimedia Large (similar to Apple's large move trailer standard 480x272) (480x270 : 75%(640x360))
320x240	4:3	Multimedia Large
320x180	16:9	Multimedia Large / Wide Screen
240x180	4:3	Multimedia Small
160x120	4:3	Thumbnail
1600x1200	4:3	Computer Display
1280x1024	4:3	Computer Display
1152x870	4:3	Computer Display
1024x768	4:3	Computer Display
800x600	4:3	Computer Display

Aspect Ratio

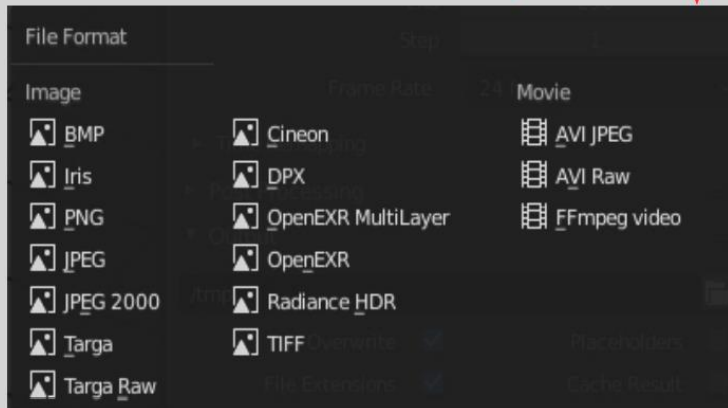
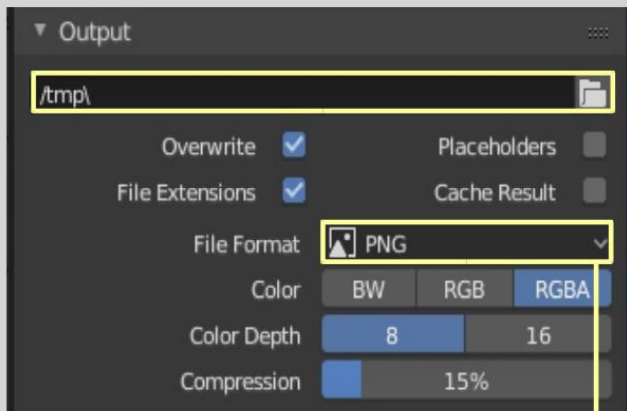
The ratio between the length and width of video images. **NTSC**, **PAL**, and **Secam** formats use a **4:3** aspect ratio. Newer, more advanced formations such as **HDTV** (High Definition Television) use a much wider aspect ratio of **16:9**.

- Television is 4:3
- Widescreen TV 16:9
- 35mm Film 1.85:1
- 70mm Film 2.0:1

The aspect ratio refers to the shape of the pixels. The default ratio X:1.000, Y:1.000 (1:1) is for computer monitors which have square pixels. TV screens have rectangular pixels so you have to set a ratio for the appropriate format i.e. HDV NTSC 1080p for America the ratio is 4:3 and HDV PAL 1080p for Europe is also 4:3 but TV PAL 16.9 the ratio is 16:11.

Blender/Python API

Render Function



Blender will save your render in a variety of file formats. The default format is **PNG** (Portable Network Graphics). Where you see this in the Output Tab is a selection menu for choosing alternative formats. In the menu you will see that the options are in two categories: **Image** and **Movie**. Image types such as PNG or JPEG produce a render of a still image in that particular file type. Selecting one of the Movie options produces a render of an animation in a compressed movie file such as AVI Raw or FFmpeg.



Blender/Python API Render Function

```
def renderToFolder(renderFolder='rendering', renderName='render', resX=800, resY=800, resPercentage=100, animation=False, frame_end = None):

    #print('renderToFolder called')
    scn = bpy.context.scene
    scn.render.engine = 'BLENDER_EEVEE_NEXT'

    scn.render.resolution_x = resX
    scn.render.resolution_y = resY
    scn.render.resolution_percentage = resPercentage
    if frame_end:
        scn.frame_end = frame_end

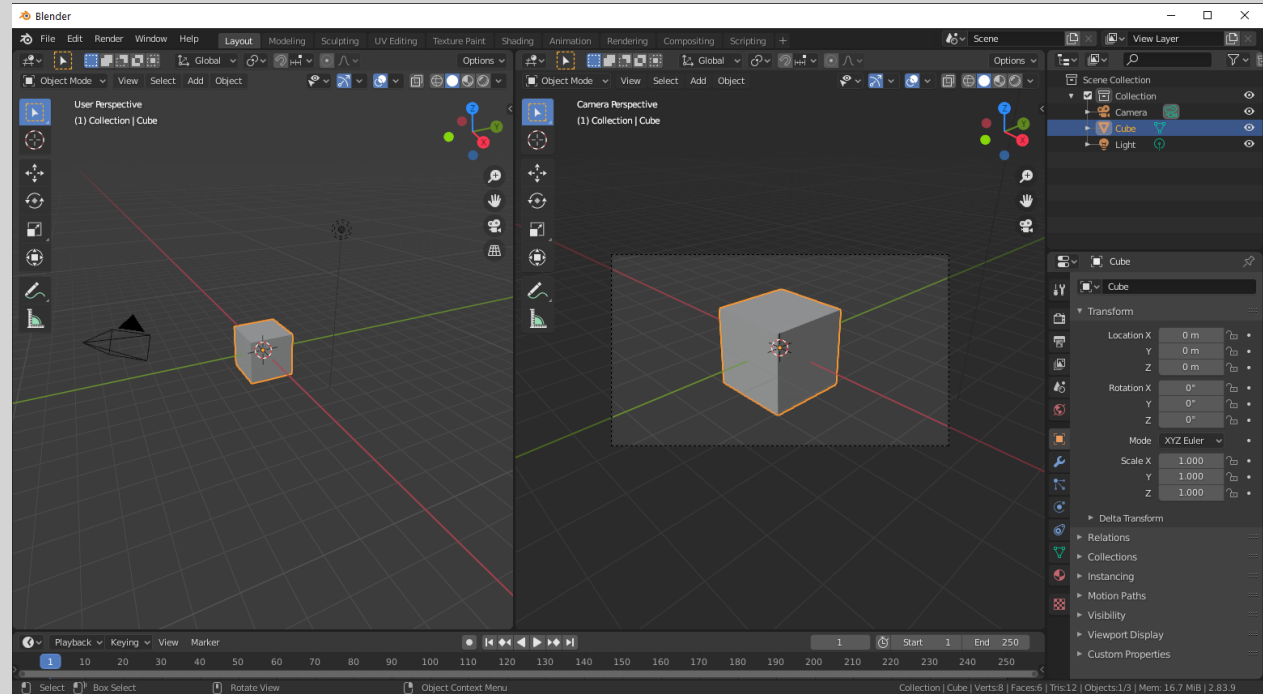
    # Specify folder to save rendering and check if it exists
    render_folder = os.path.join(os.getcwd(), renderFolder)
    if (not os.path.exists(render_folder)):
        os.mkdir(render_folder)

    if animation:
        # Render animation
        scn.render.filepath = os.path.join( render_folder, renderName)
        bpy.ops.render.render('INVOKE_DEFAULT', animation = True)
    else:
        # Render still frame
        scn.render.image_settings.file_format = 'PNG'
        scn.render.filepath = os.path.join( render_folder, renderName + '.png')
        bpy.ops.render.render('INVOKE_DEFAULT', write_still = True)
        print( renderName + '.png is Rendered')
```

Blender/Python API

Camera Tracking

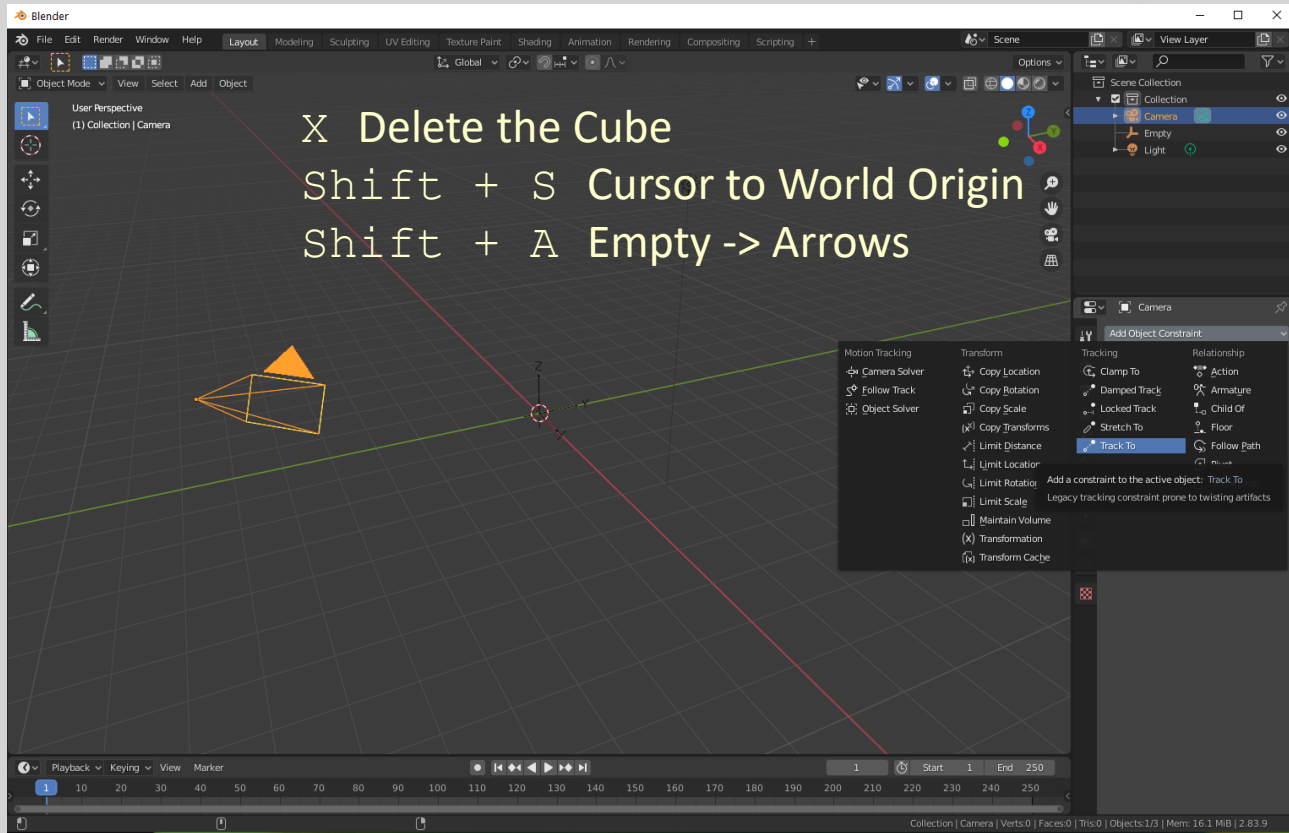
In the default Blender Scene a Camera Object is directed towards the Cube Object such that it captures the Cube in Camera View (**Num Pad 0**). When the Cube is animated to move across the Screen the Cube can move in and out of Camera View.



If you want the Cube to remain in view no matter where the Cube is in the Scene, you track the Camera to the Cube by employing the Track To Constraint

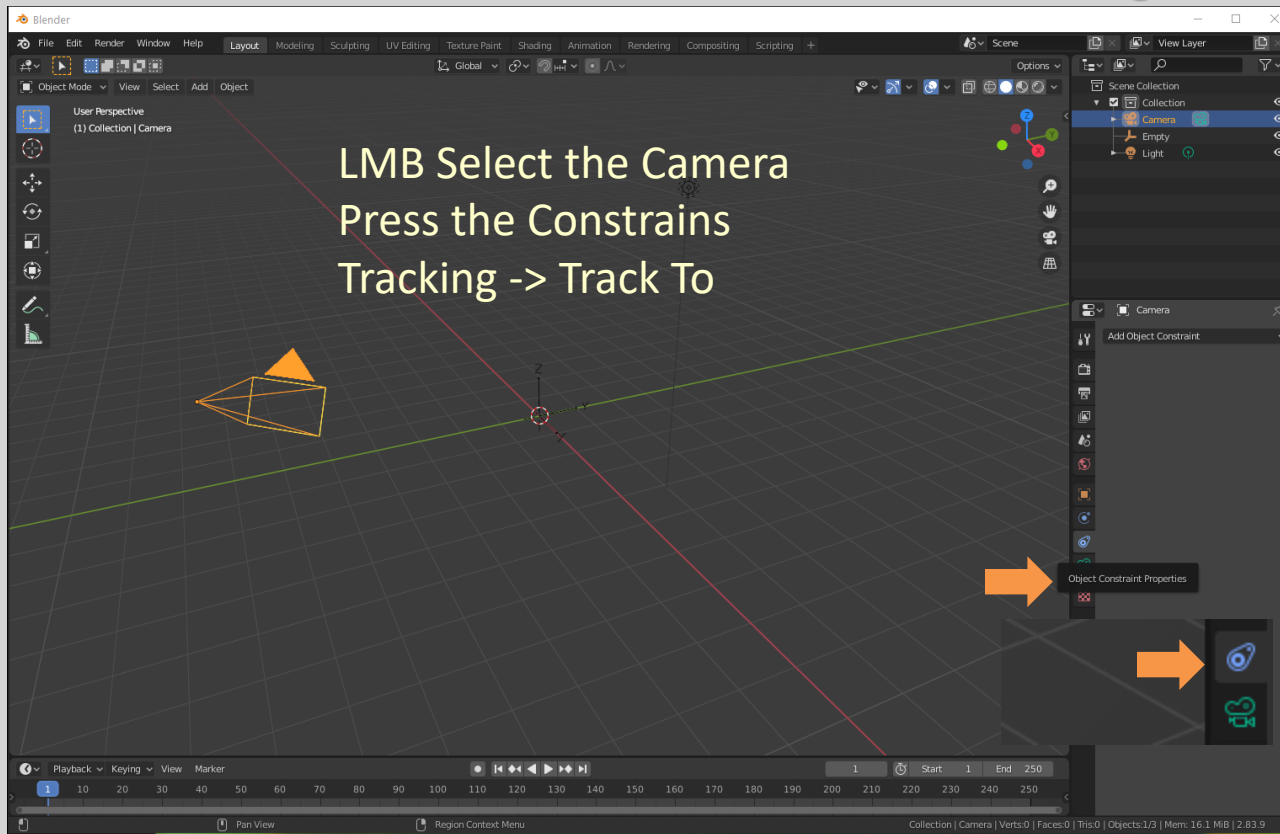


Blender/Python API Camera Tracking



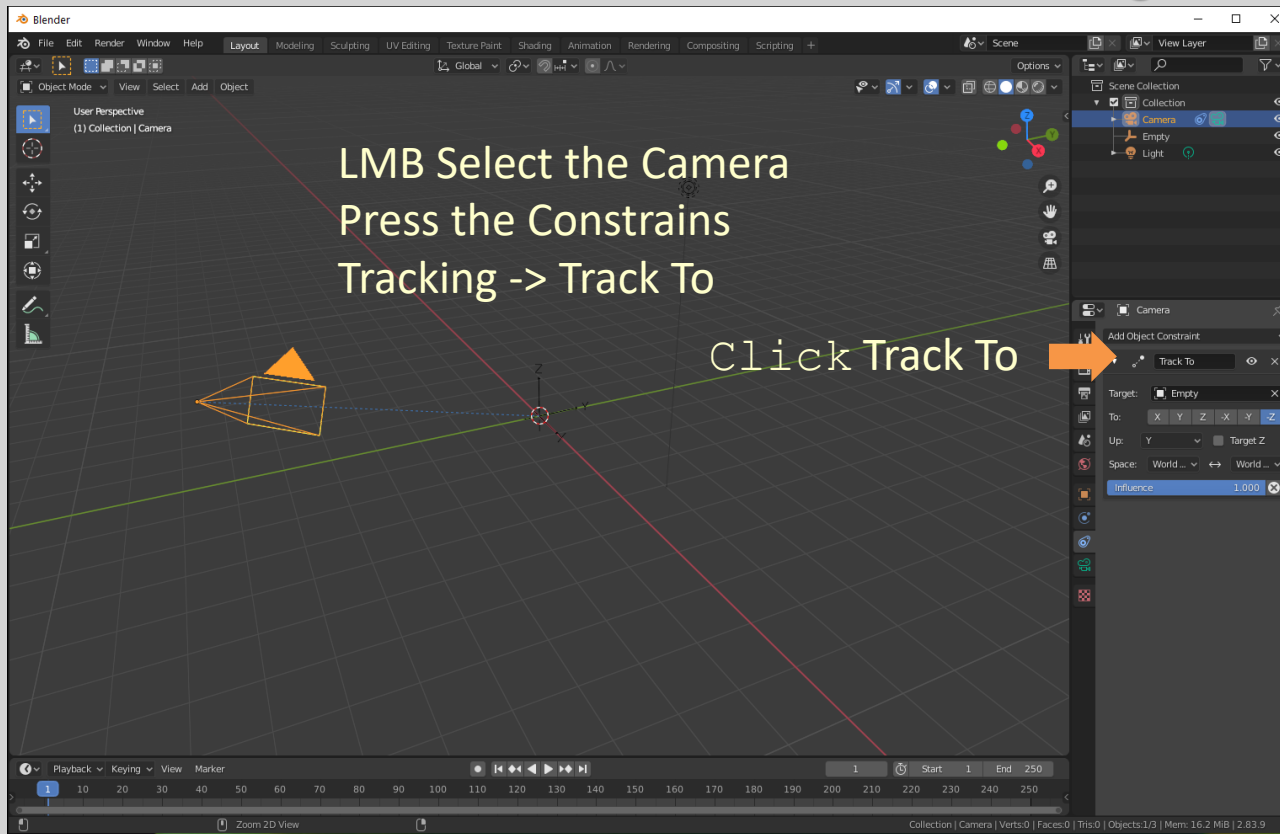
Blender/Python API

Camera Tracking



Blender/Python API

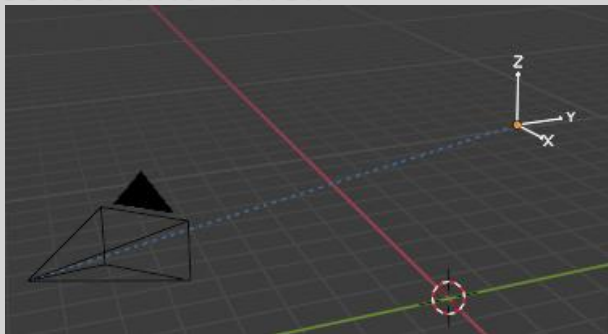
Camera Tracking



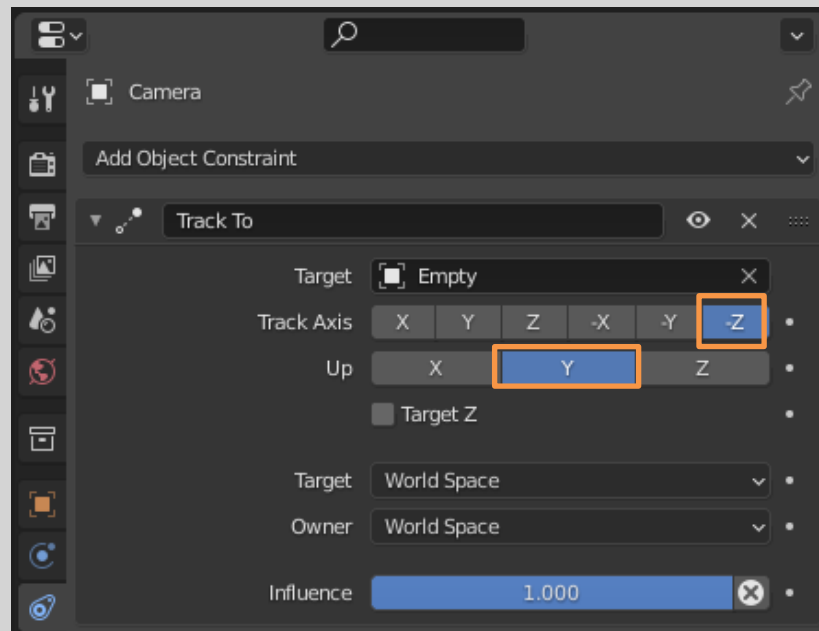


Blender/Python API Camera Tracking

There is a broken line connecting the Camera to the Empty indicating that a Constraint is applied, but you have to adjust To and Up directions in the Constraint Panel.

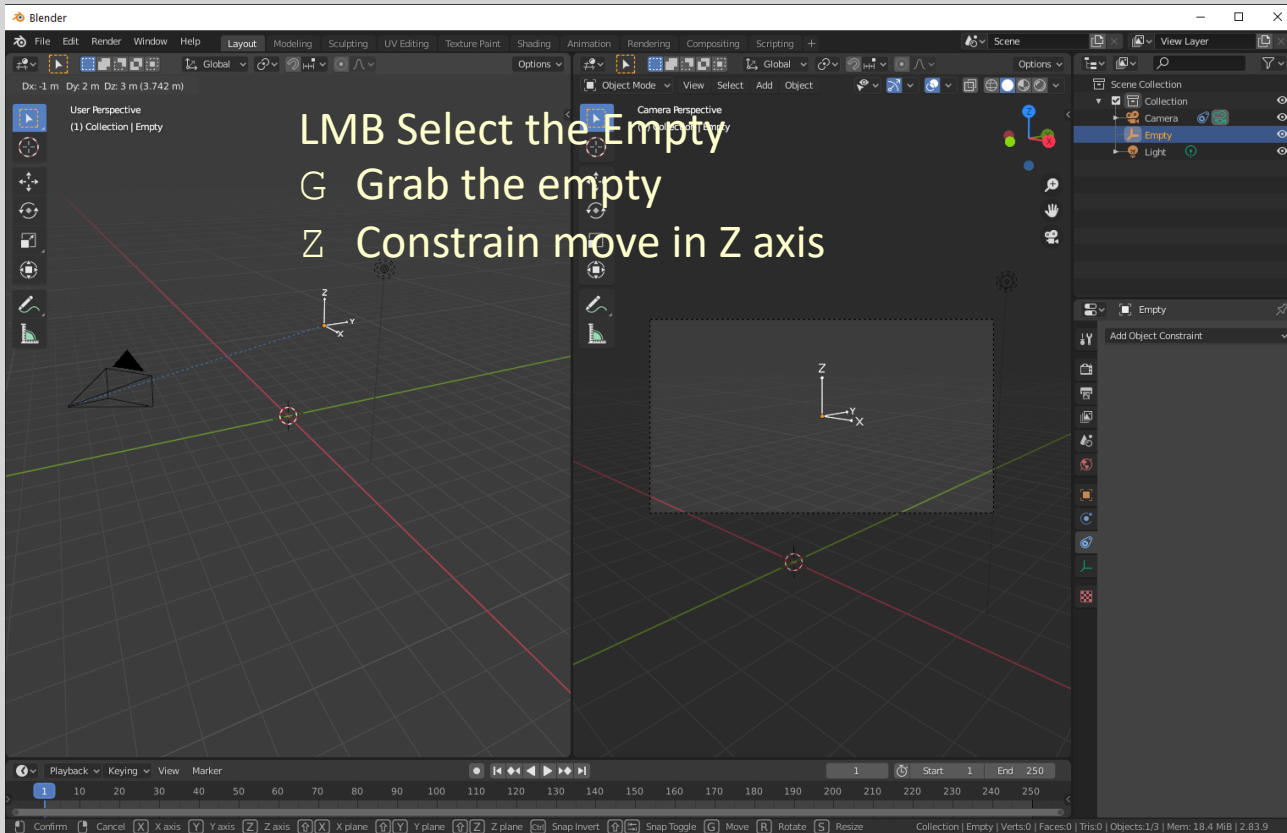


Set To as -Z and Up as Y. With the Empty animated to move in the Scene the Camera always points to the Empty.





Blender/Python API Camera Tracking





Blender/Python API Tracking

```
def trackToConstraint(obj, target):  
    constraint = obj.constraints.new('TRACK_TO')  
    constraint.target = target  
    constraint.track_axis = 'TRACK_NEGATIVE_Z'  
    #constraint.track_axis = 'TRACK_Z'  
    constraint.up_axis = 'UP_Y'  
    #constraint.owner_space = 'LOCAL'  
    #constraint.target_space = 'LOCAL'  
  
    return constraint  
  
def target(origin=(0,0,0)):  
    tar = bpy.data.objects.new('Target', None)  
    bpy.context.scene.objects.link(tar)  
    tar.location = origin  
  
    return tar
```



Blender/Python API Camera Tracking

```
def camera(origin, target=None, lens=35, clip_start=0.1, clip_end=200, type='PERSP', ortho_scale=6):  
    # Create object and camera  
    bpy.ops.object.camera_add(location=origin)  
    obj = bpy.context.object  
    obj.data.lens = lens  
    obj.data.clip_start = clip_start  
    obj.data.clip_end = clip_end  
    # 'PERSP', 'ORTHO', 'PANO'  
    obj.data.type = type  
    if type == 'ORTHO':  
        obj.data.ortho_scale = ortho_scale  
  
    if target:  
        trackToConstraint(obj, target)  
  
    return obj
```

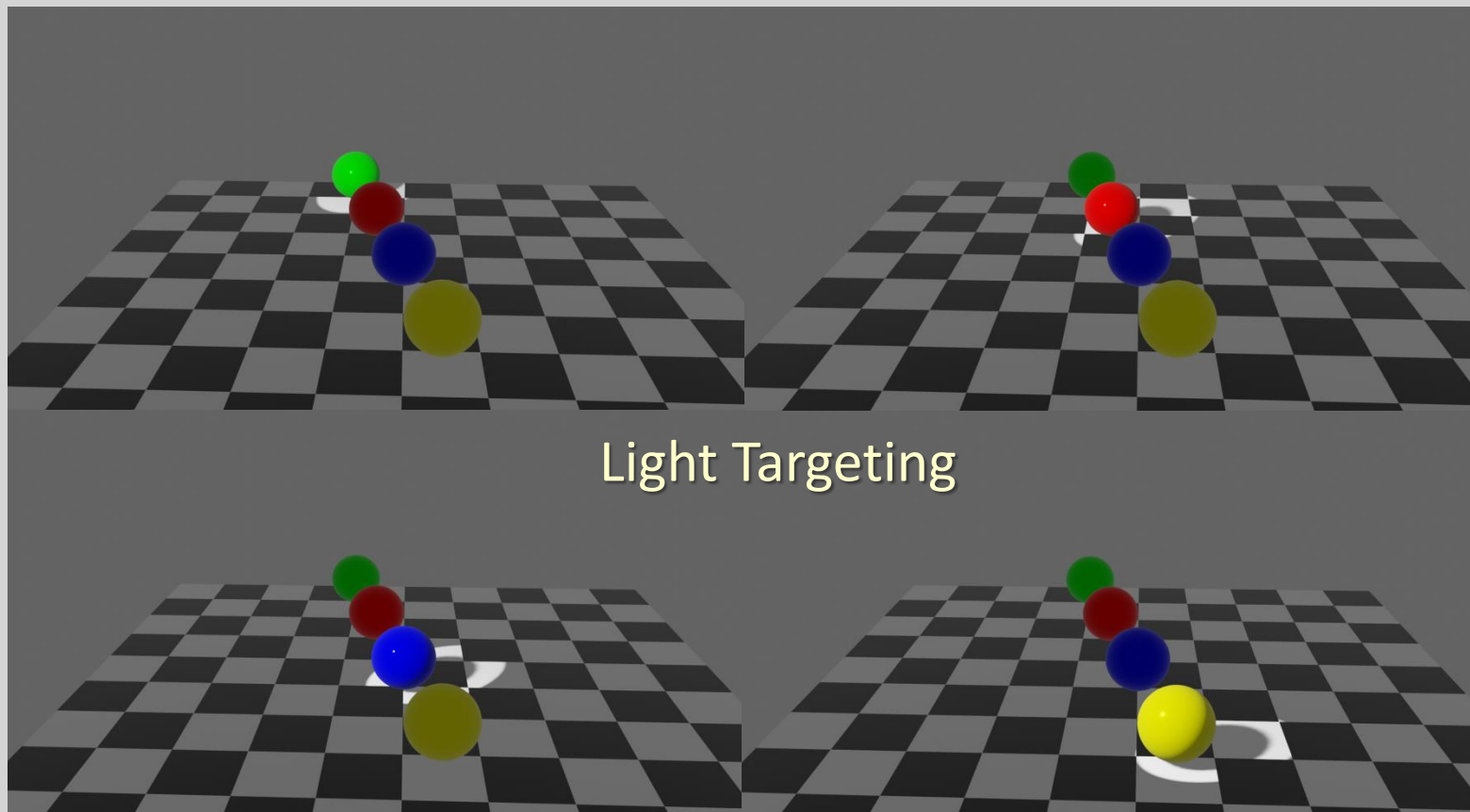


Blender/Python API Lamp Tracking

```
def light(origin, target=None, lamptype='POINT', energy=1, color=(1,1,1)):  
    # Lamp types: 'POINT', 'SUN', 'SPOT', 'HEMI', 'AREA'  
    # print('createLamp called')  
    bpy.ops.object.light_add(type=lamptype, location = origin)  
  
    obj = bpy.context.object  
    obj.data.energy = energy  
    obj.data.color = color  
    if target:  
        trackToConstraint(obj, target)  
  
    return obj
```

Blender/Python API

Homework I



Blender/Python API

Homework II

Spot Light Area Sizing

