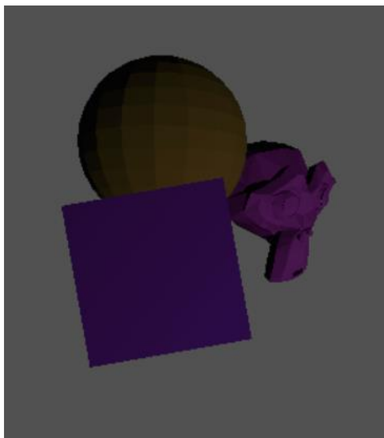
VR Rendering (Real)

Faye Nguyen

Recap



- Implement IPD
- Distort using Brown-Conrad’s Distortion Model

$$X_d = X_u \left(k_{offset} + k_0 r^0 + k_1 r^1 + \cdots + k_6 r^6\right)$$

$$Y_d = Y_u \left(k_{offset} + k_0 r^0 + k_1 r^1 + \cdots + k_6 r^6\right)$$

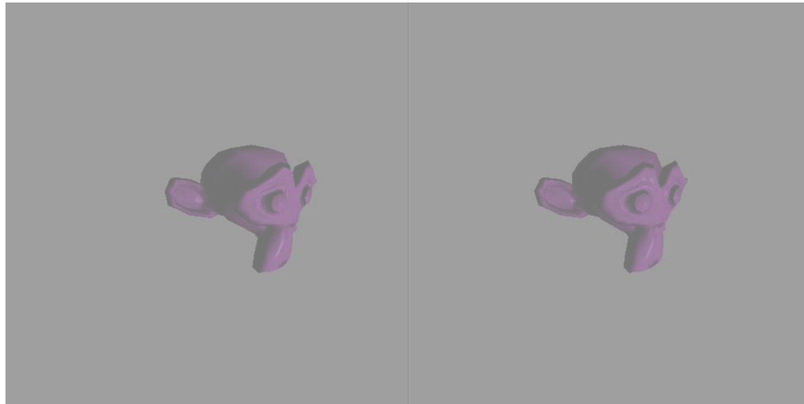
(x_d, y_d) = distorted image point as projected on image plane using specified lens,
(x_u, y_u) = undistorted image point as projected by an ideal pin-hole camera,

IPD Methods

- Two cameras with an X-position offset (to mimic eyes)
- **Trouble**: No sense of scale from millimeters to numbers in renderer. I have to guess the distance.
- In code, rendered the two images on top of each other, then one result is shoved by an offset-amount (the right camera's render)

IPD Methods

- Two cameras with an X-position offset (to mimic eyes)
- **Trouble**: No sense of scale from millimeters to numbers in renderer. I have to guess the distance.
- In code, rendered the two images on top of each other, then one result is shoved by an offset-amount (the right camera’s render)



Eyesight Class

```python
'''
Sets up L/R Cams using original position of camera
'''
def setup_cams(self, IPD):
    # X translation
    transformLeft = Transform()
    transformLeft.set_position(-IPD/2, 0, 0)
    transformRight = Transform()
    transformRight.set_position(IPD/2, 0, 0)

    # Apply to L camera
    cam_L = PerspectiveCamera(self.left, self.right, self.bottom, self.top, self.near, self.far)
    trans_L = self.transform.get_position() + transformLeft.get_position()
    cam_L.transform.set_position(trans_L[0], trans_L[1], trans_L[2])

    # Apply to R camera
    cam_R = PerspectiveCamera(self.left, self.right, self.bottom, self.top, self.near, self.far)
    trans_R = self.transform.get_position() + transformRight.get_position()
    cam_R.transform.set_position(trans_R[0], trans_R[1], trans_R[2])

    return (cam_L, cam_R)
```

Render Loop Addition

```
def render_VR(self, shading, bg_color, ambient_light):
    image_buffer = np.full((2*self.screen.width, self.screen.height, 3), bg_color)
    offset = 0 # For loading L/R side of screen
    counter = 1

    ''' Per Camera Lighting'''
    for camera in self.eyesight.cams:
        depth_buffer = np.full([2*self.screen.width, self.screen.height, 1], -np.inf, dtype=float)
        self.render_grids(offset, image_buffer)

        # Let users know about our progress
        loading_statement = "Loading Left Eye" if (offset == 0) else "Loading Right Eye"
        print(loading_statement)

        # Iterate over vertices (vertex shader)
        for mesh in self.meshes:
            print("Loading mesh", counter, "of", len(self.meshes * 2))
            counter += 1
```

Distortion Methods

- Couldn’t find coefficient values online. Had to use profiler to **manually guess the correct coefficients**. Probably not right, but close enough.
- Distortion coefficients from profiler were on a **totally different scale**
    - and **not negative** (negative coefficients are used for barrel distortions)

Tray to lens-center distance (mm)

35.0

Distortion coefficients

k_1 0.260

k_2 0.390

$$X_d = X_u\,(k_{offset} + k_0 r^0 + k_1 r^1 + \cdots + k_6 r^6)$$

$$Y_d = Y_u\,(k_{offset} + k_0 r^0 + k_1 r^1 + \cdots + k_6 r^6)$$

(x_d, y_d) = distorted image point as projected on image plane using specified lens,
(x_u, y_u) = undistorted image point as projected by an ideal pin-hole camera,

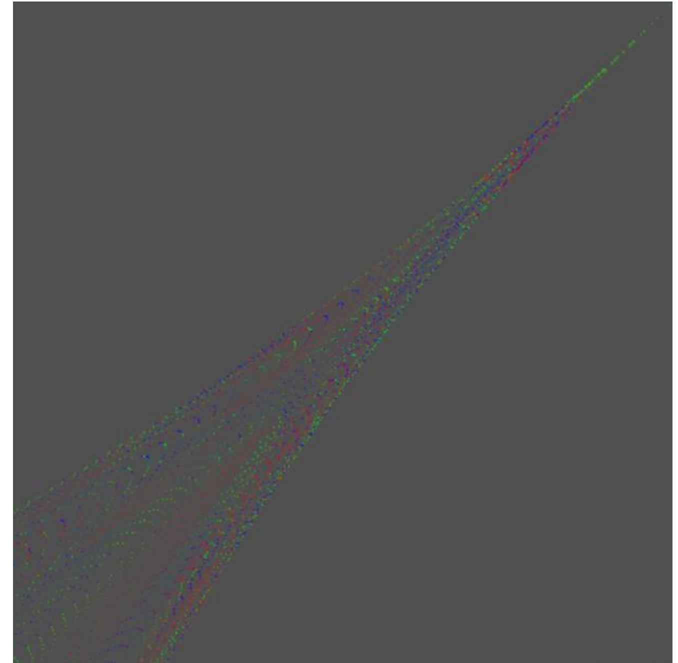
```
K_OFFSET = 0 # Value 0 for specifically for Vive SDK
KTH = [-0.00000260, -0.00000390, 0, 0, 0, 0]    # Di
# KTH = [0.0260, 0.0390, 0, 0, 0, 0]    # Distortion
```

Distortion (2) Method

- Couldn't make Brown-Conrady model work for some reason
- Used the **Reverse Distortion model** instead
  - Required only 1 coefficient to make things convenient
  - However, less accurate. But good enough

$$x_{\mathrm{d}} = x_{\mathrm{c}} + \frac{x_{\mathrm{u}} - x_{\mathrm{c}}}{2K_1 r_u^2}\left(1 - \sqrt{1 - 4K_1 r_u^2}\right)$$

$$y_{\mathrm{d}} = y_{\mathrm{c}} + \frac{y_{\mathrm{u}} - y_{\mathrm{c}}}{2K_1 r_u^2}\left(1 - \sqrt{1 - 4K_1 r_u^2}\right),$$



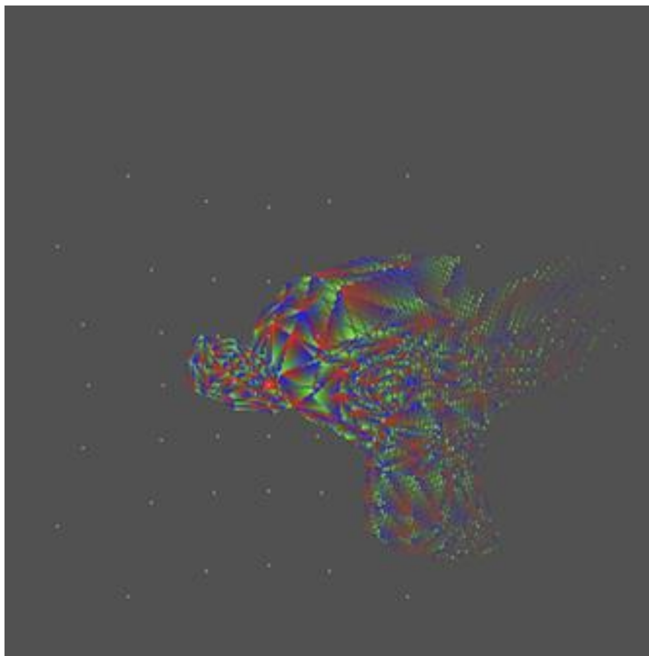
Rendering Addition

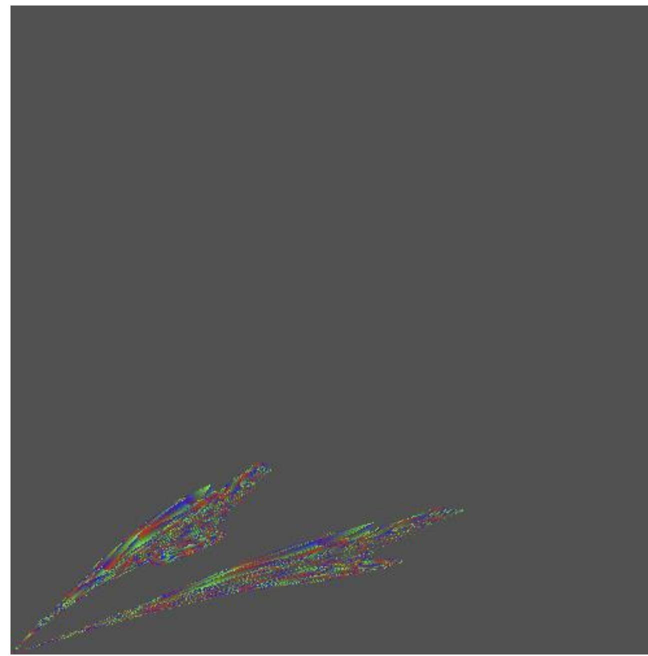
```
try:
    (x_d, y_d) = self.reverse_distortion(x + offset, y, MIDPOINTS[0] + offset, MIDPOINTS[1])
    image_buffer[x_d, y_d] = [
        int(RED[0] * irradiance[0]),
        int(GREEN[1] * irradiance[1]),
        int(BLUE[2] * irradiance[2])
    ]

except:
    image_buffer[x + offset,y] = [0, 255, 0]
    print("Lighting Equation Error")
```

Distortion Equation

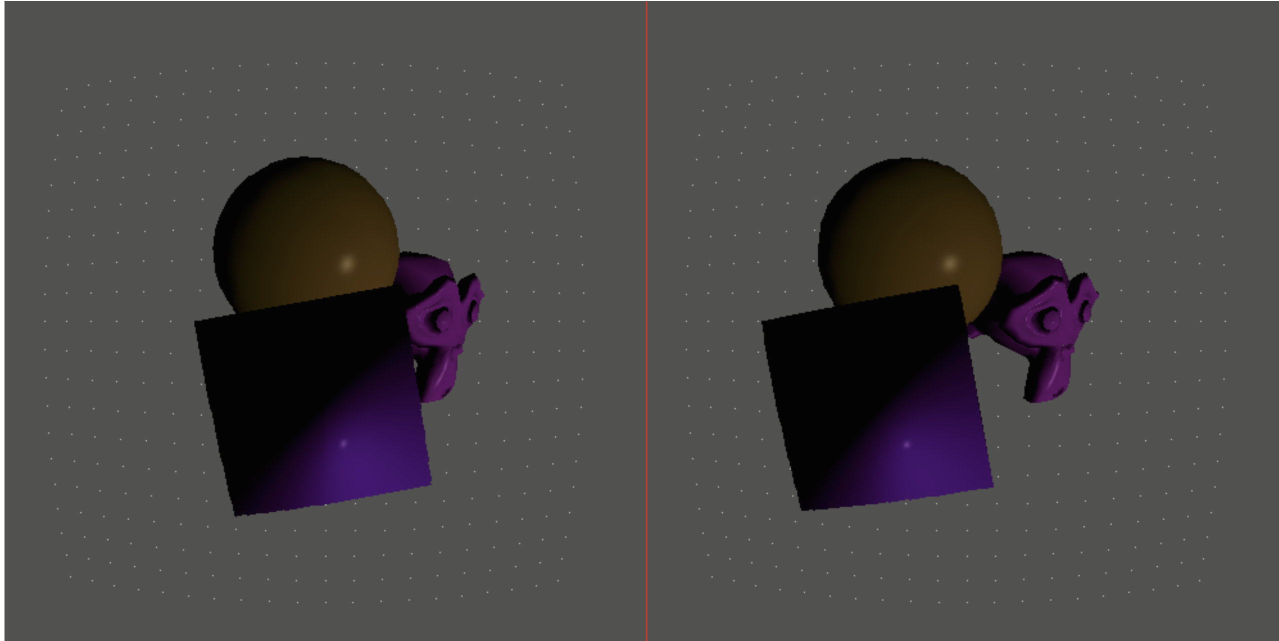
```
def reverse_distortion(self, x, y, x_c, y_c):
    dist_x = (x - x_c)
    dist_y = (y - y_c)

    r = np.sqrt((dist_x ** 2) + (dist_y ** 2)) # euclidian distance between undistorted point & distortion center
    if (r == 0.0):
        r = 0.0000000001 # prevent divide by 0
    term_2x = dist_x / (2 * KTH[0] * (r ** 2))
    term_2y = dist_y / (2 * KTH[0] * (r ** 2))
    term_3 = (1 - np.sqrt(1 - (4 * KTH[0] * (r ** 2))))

    x_d = x_c + (term_2x * term_3)
    y_d = y_c + (term_2y * term_3)

    return (int(x_d), int(y_d))
```

Funny Renders





Final Deliverable



I actually added in the grid for debugging, but it’s also good to verify that the lines are horizontal in the viewer. It also makes it hurt less (to me)

Final Remarks: What could be better

IPD

- Better IPD adjustment

Distortion

- Get exact coefficients from Google API’s get_distortion_coefficients()
- Factor in other variables (screen-to-lens, tray-to-lens)

Google Cardboard

- super glue

Phone

- screen protector