University of BRISTOL

COMS 30115

Rendering: Raytracing

Carl Henrik Ek - carlhenrik.ek@bristol.ac.uk

February 8, 2019

http://www.carlhenrik.com

Introduction

Info

- Coursework
  - Find a partner

Info

- Coursework
  - Find a partner
- Reddit
  - its reddit, cat gifs are OK etc.
  - if you found an awesome graphics resource etc.

Info

- Coursework
  - Find a partner
- Reddit
  - its reddit, cat gifs are OK etc.
  - if you found an awesome graphics resource etc.
  - its annonomous, tell me if I'm an idiot and the lecture/coursework being stupid

Today

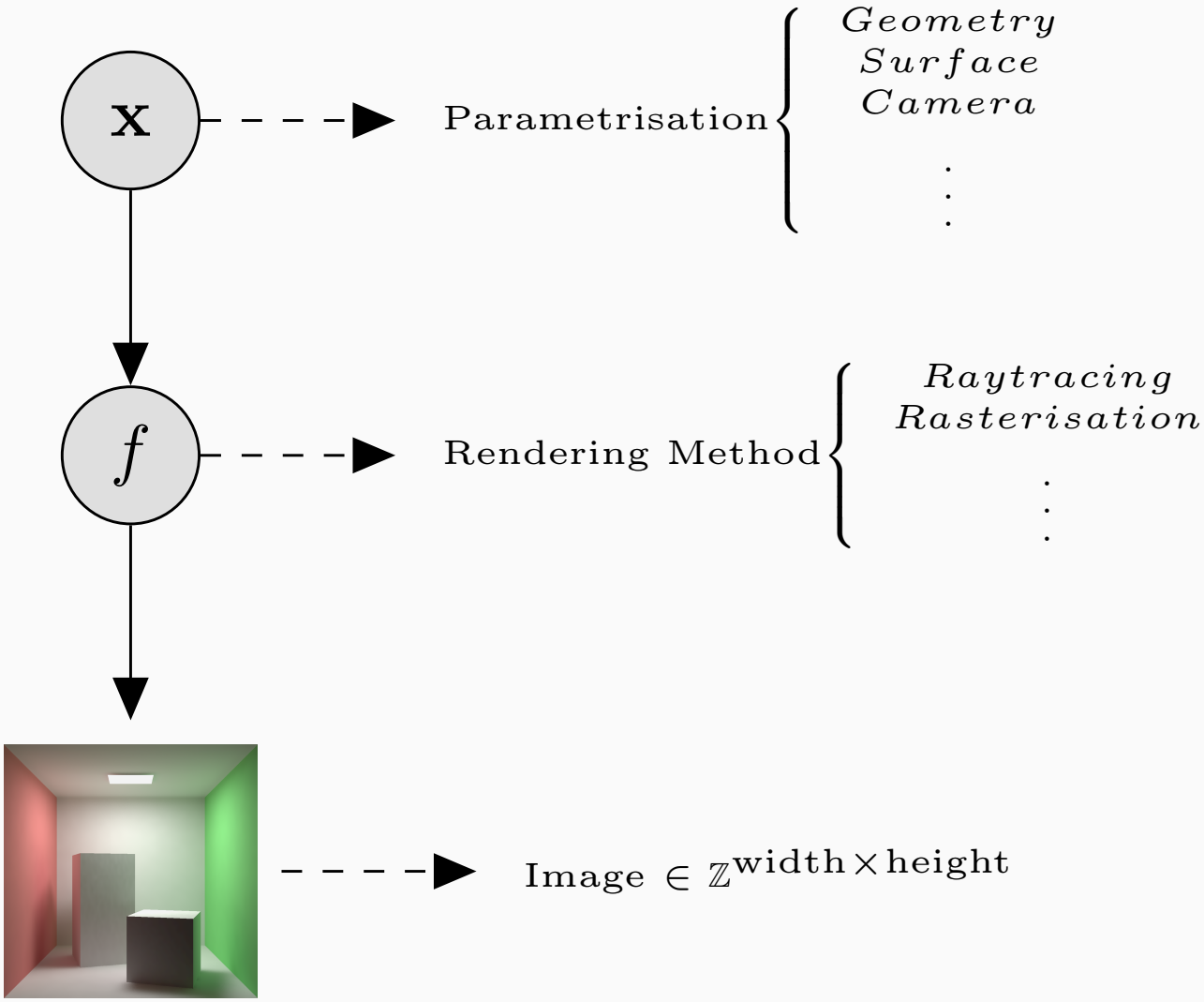
- Generative Model and Assumptions
- First rendering method
  - Raytracing/Raycasting
  - Cameras
  - Intersection calculations
- After today you should be able to start with Lab 1 (sorry)

Scratchapixel

- Introduction to Ray Tracing: a Simple Method for Creating 3D Images
- An Overview of the Ray-Tracing Rendering Technique

Generative Model

Generative Model



Images

- Rectangular grid of elements
- Parametrization
  - Location (x, y)
  - Colour p

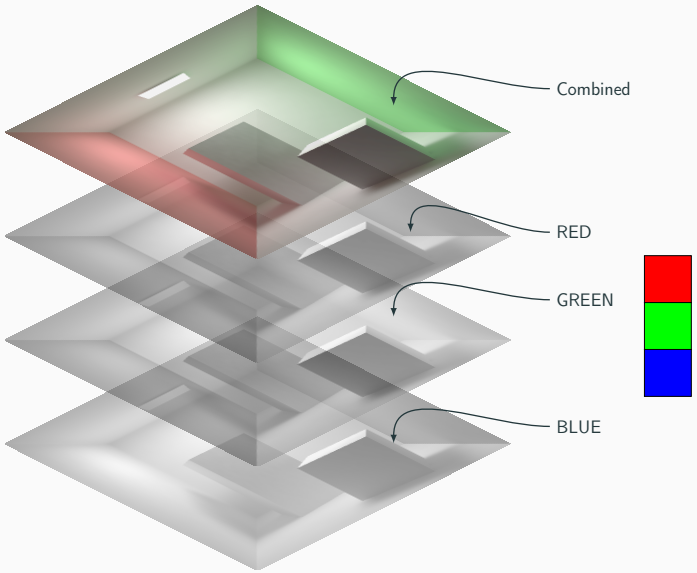
Images

Encoding

- Indexed Images
  - GIF, PNG have this format
  - Palette (look-up)
  - Efficient if few colours in image
  - Older hardware implemented this
- (True) colour Images
  - Directly stores Colour values
  - Simpler, not limited to palette size
  - Less efficient for low colour images

Pixels

- Colour
  - RGB - `RED`, `GREEN`, `BLUE`
  - HSV - `HUE`, `SATURATION`, `VALUE`
- Nr of bits known as the depth
- What is the effect of the parametrisation?
- Similarity



Pixel Parametrisation

The Dress that broke the Internet[1]

[1] https://en.wikipedia.org/wiki/The_dress

Colour

- Psychological
  - colours encode meaning
  - meaning is highly subjective
- Physiological
  - internal processes of colour
  - optical illusions etc.

Colour

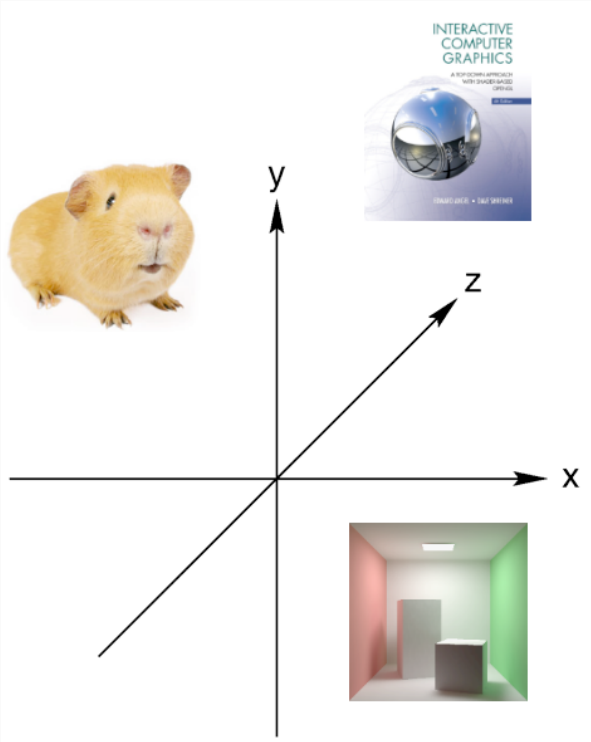
- Psychological
  - colours encode meaning
  - meaning is highly subjective
- Physiological
  - internal processes of colour
  - optical illusions etc.
- *an image is interpreted by us, we never see the pixels*

Colour like a coder

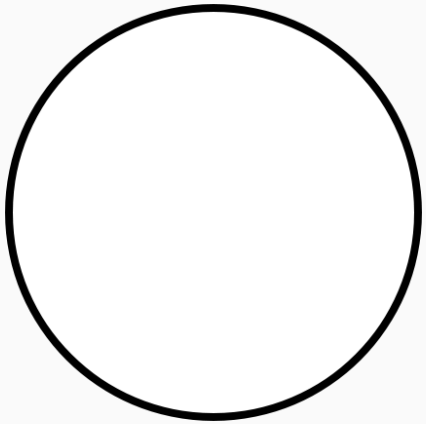
Red is not a colour it is `0x00ff0000`

Image Parametrisation

- RGB pixel $p \in \mathbb{Z}^3$
- Image as a point in a vector space
- $I \in (\mathbb{R}^3)^{width \times height}$
- Typical Image size
  $1920 \times 1080 \times 3 = 6220800$



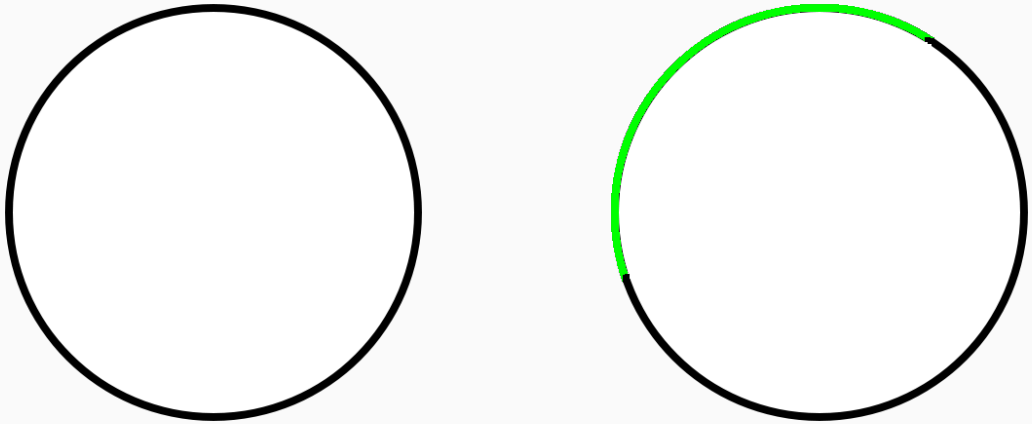
Manifolds



"In mathematics, a manifold is a topological space that near each point resembles Euclidean space"[2]

[2] http://en.wikipedia.org/wiki/Manifold

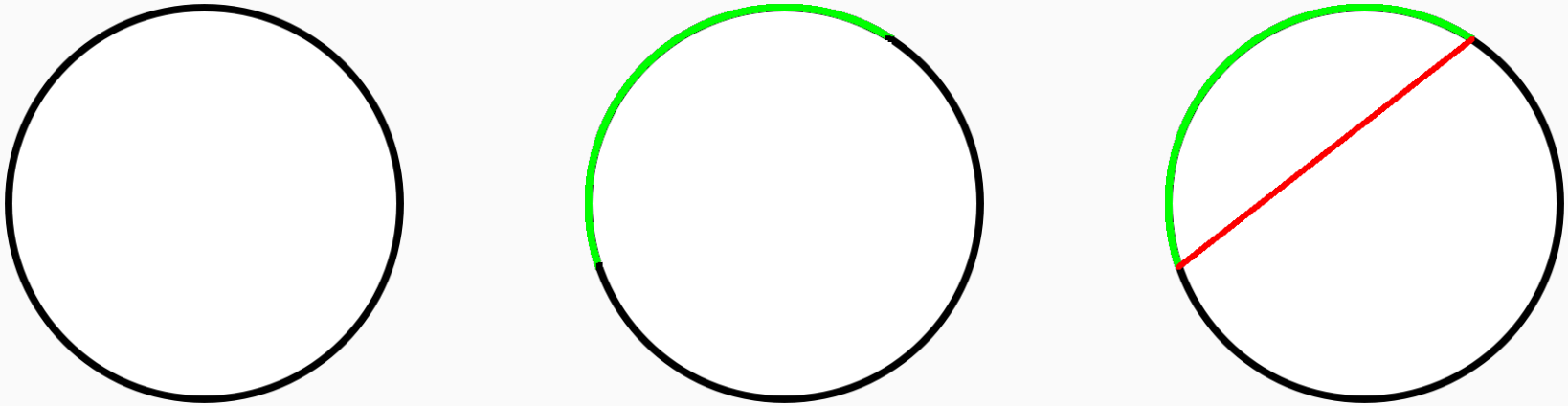
Manifolds



"In mathematics, a manifold is a topological space that near each point resembles Euclidean space"[2]

[2] http://en.wikipedia.org/wiki/Manifold

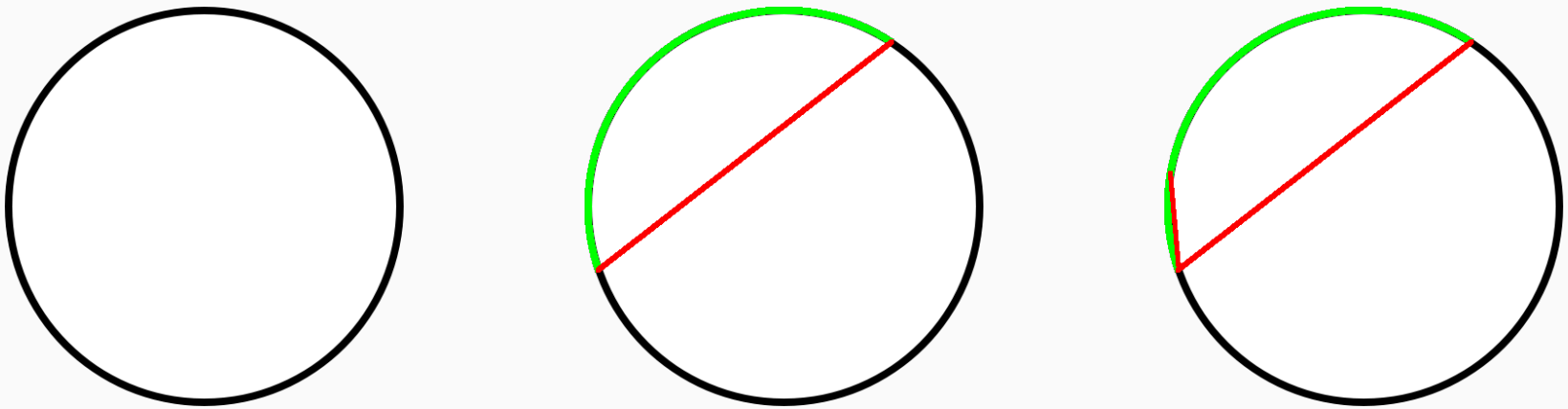
Manifolds



"In mathematics, a manifold is a topological space that near each point resembles Euclidean space"[2]

[2] http://en.wikipedia.org/wiki/Manifold

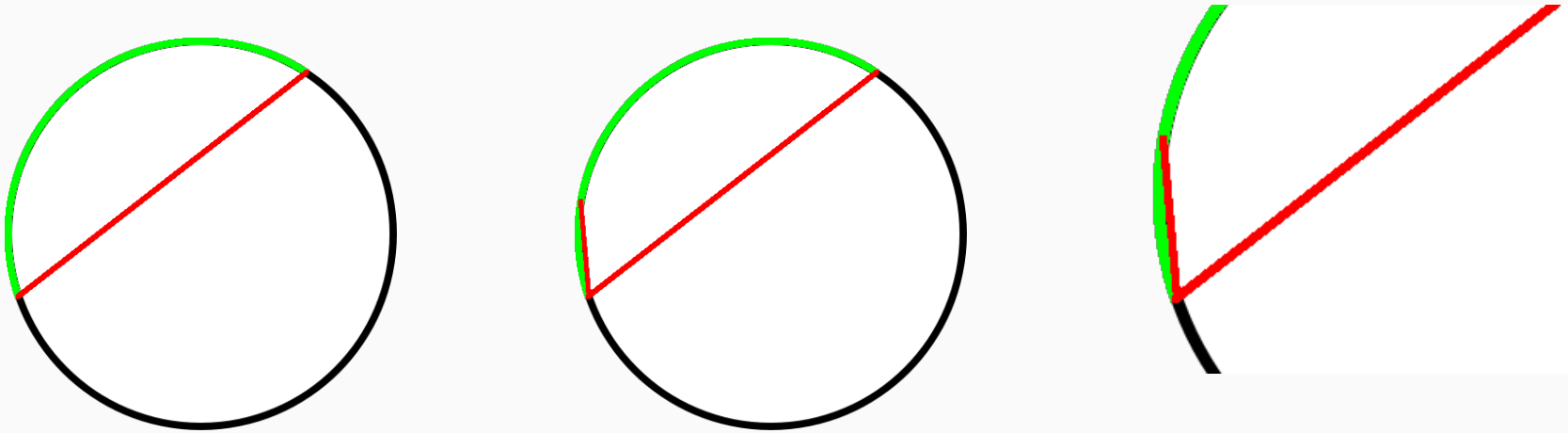
Manifolds



"In mathematics, a manifold is a topological space that near each point resembles Euclidean space"[2]

[2] http://en.wikipedia.org/wiki/Manifold

Manifolds



"In mathematics, a manifold is a topological space that near each point resembles Euclidean space"[2]

[2] http://en.wikipedia.org/wiki/Manifold

Parametrisation

- Image space is a manifold in space of matrices with positive values
- Degrees of freedom is coordinates along manifold
    - Each parameter configuration generates valid image
- *how can we find these parametrers?*

Parametrisation

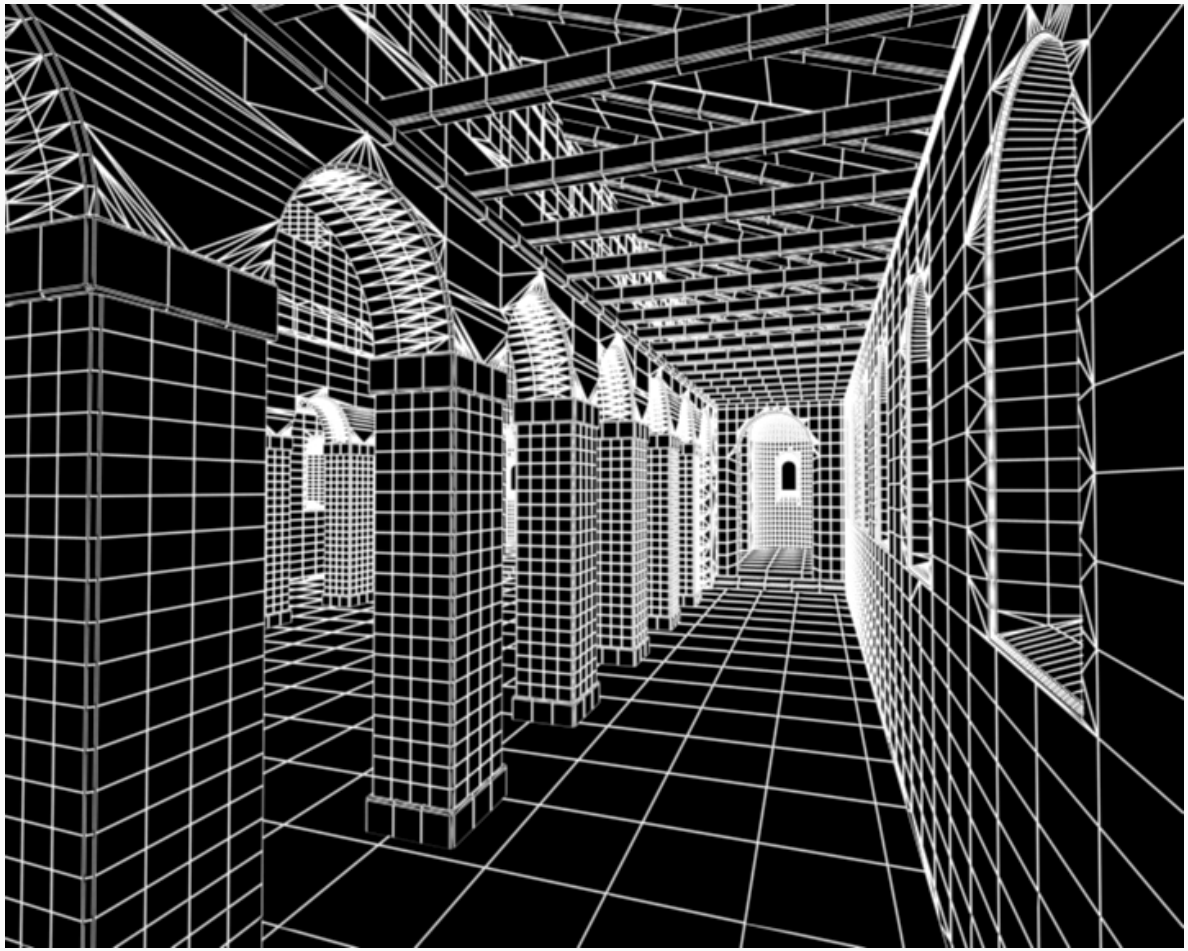
- Image space is a manifold in space of matrices with positive values
- Degrees of freedom is coordinates along manifold
  - Each parameter configuration generates valid image
- *how can we find these parametrers?*
  - Light If there is no light all environments look the same
  - Geometry Light bounces of surfaces to eye/camera, i.e. we see surface
  - Surface material, texture, colour etc. alters light
  - Imaging device Camera/eye characteristics

Parametrisation

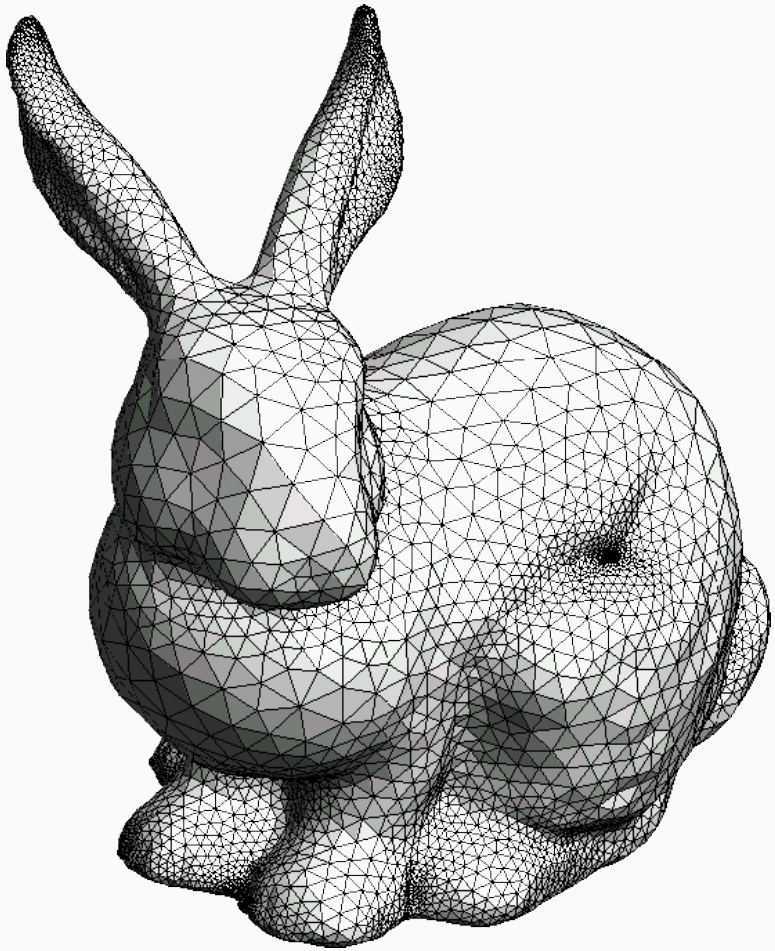
- Image space is a manifold in space of matrices with positive values
- Degrees of freedom is coordinates along manifold
  - Each parameter configuration generates valid image
- *how can we find these parametrers?*
  - Light If there is no light all environments look the same
  - Geometry Light bounces of surfaces to eye/camera, i.e. we see surface
  - Surface material, texture, colour etc. alters light
  - Imaging device Camera/eye characteristics
- We will study both the parametrisation and the generative mapping

Parametrisation

Geometry

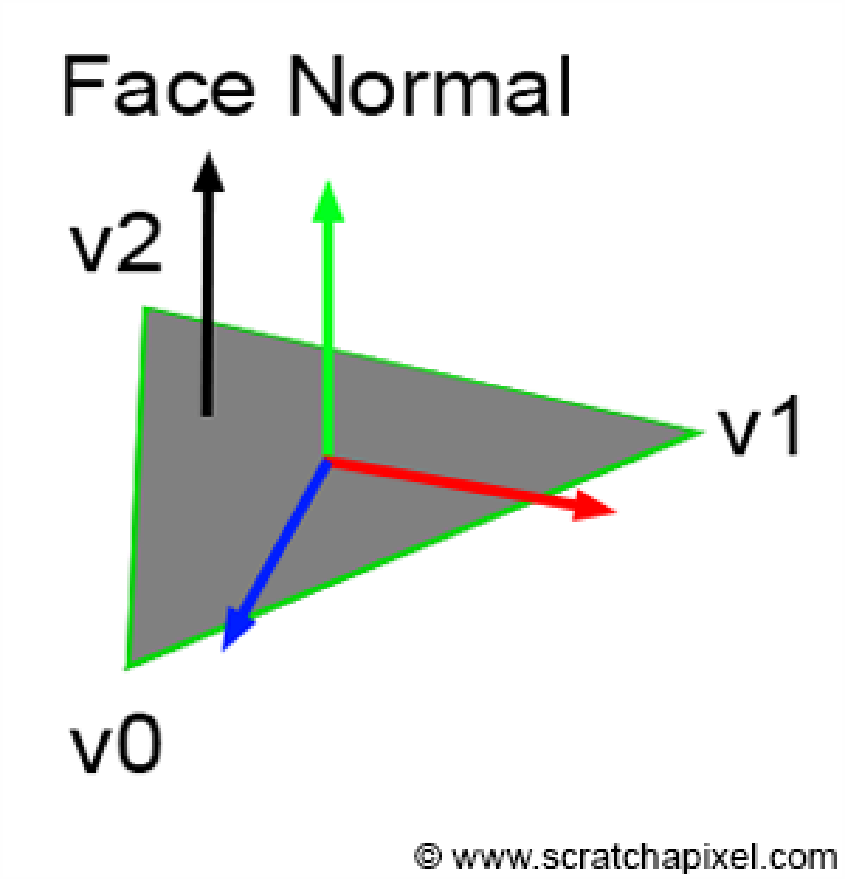


Meshes[3]

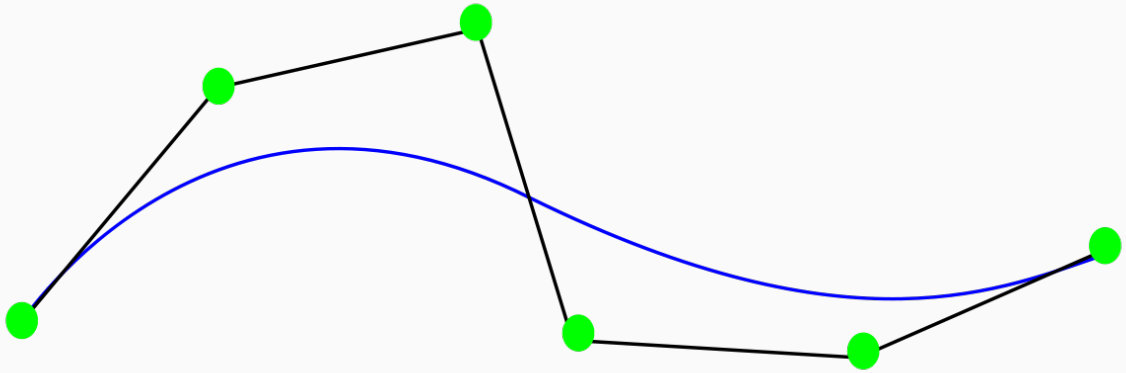


[3] https://en.wikipedia.org/wiki/Stanford_bunny

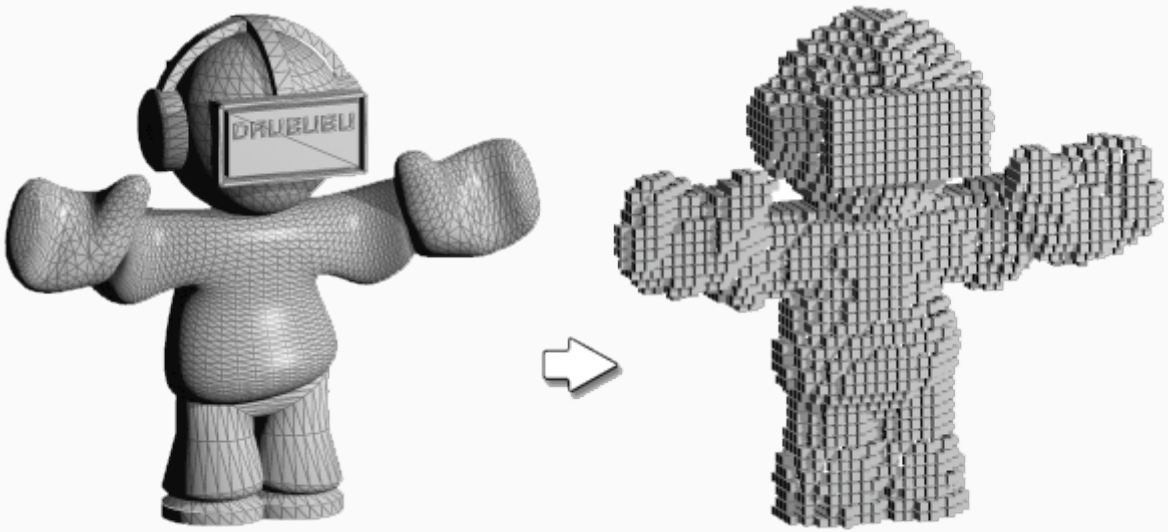
Triangles



Non-planar Geometry



Voxels[4]



[4] https://www.gamersnexus.net/gg/762-voxels-vs-vertexes-in-games

Assumptions

- The world is locally flat
- Parametrise each object as a set of triangles
- Each object is just an empty hull
- Everything has a fixed resolution

Illumination

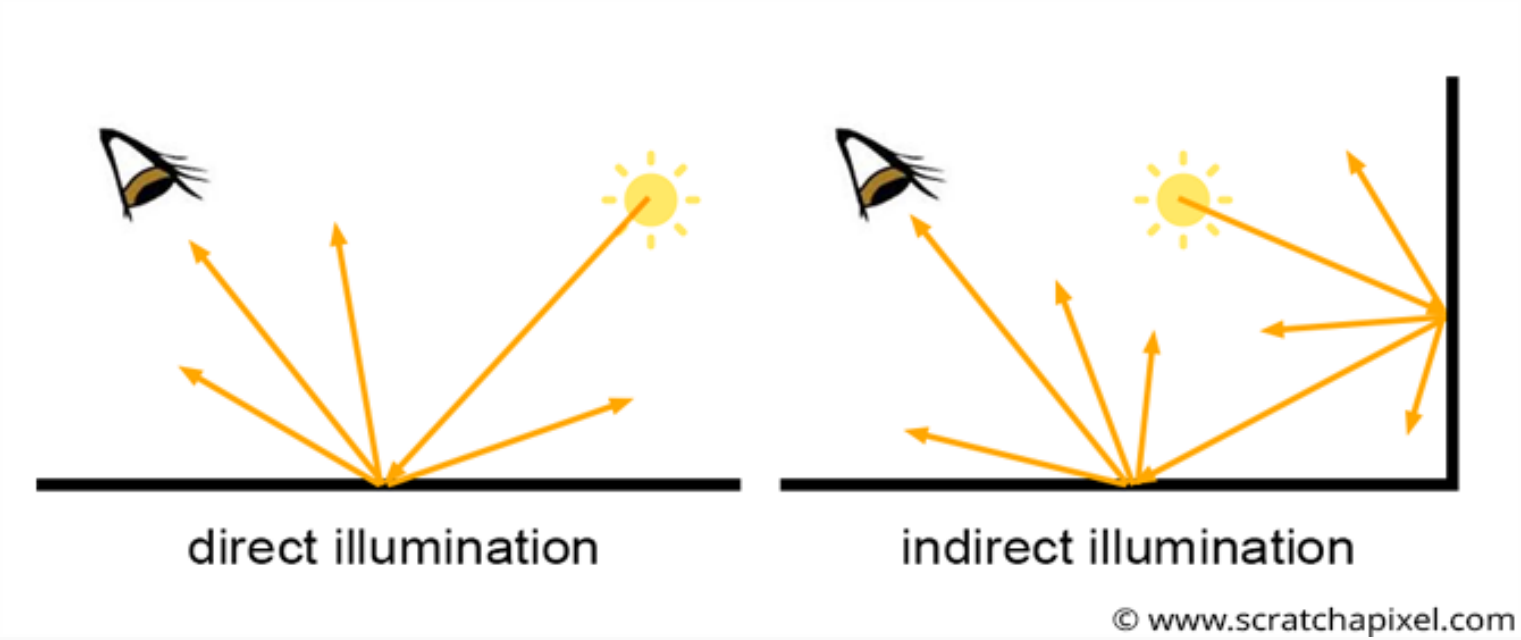
Light

- generated from lightsources (i.e. sun, lamps)
- "bounces" around in the world
- surface visible when hit by light

direct source $\Rightarrow$ surface

indirect surface $\Rightarrow$ surface

Illumination



Transport Problem

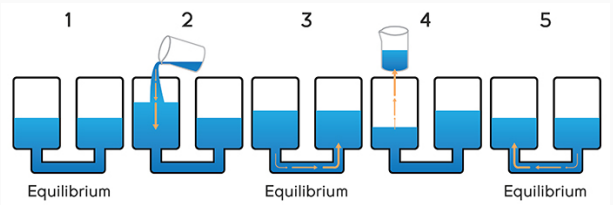
- Illuminating/Rendering $\Rightarrow$ compute interaction between incident light and surface
- Compute transportation

Transport Problem

- Illuminating/Rendering $\Rightarrow$ compute interaction between incident light and surface
- Compute transportation
- Conservation principle
  - nothing is created except for at the source

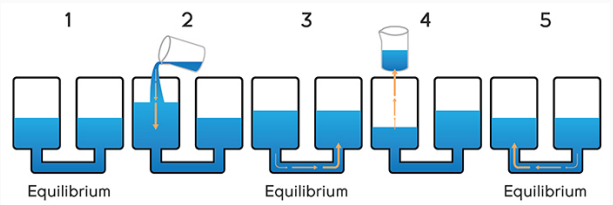
Transport Problem

- Illuminating/Rendering $\Rightarrow$ compute interaction between incident light and surface
- Compute transportation
- Conservation principle
  - nothing is created except for at the source
- light is *very* fast



Transport Problem

- Illuminating/Rendering $\Rightarrow$ compute interaction between incident light and surface
- Compute transportation
- Conservation principle
  - nothing is created except for at the source
- light is *very* fast
- Assume that it is at equlibrium



Rendering Equation [5]

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \rightarrow \Theta) L(x \leftarrow \Psi) cos(\mathbf{n}_x, \Psi) d\omega_\Psi$$

- Formulates transport problem of light
- Encapsulates "all" rendering techniques
- Intractable

[5] Kajiya, J. T. (1986). The rendering equation. In , ACM Siggraph Computer Graphics (pp. 143–150)

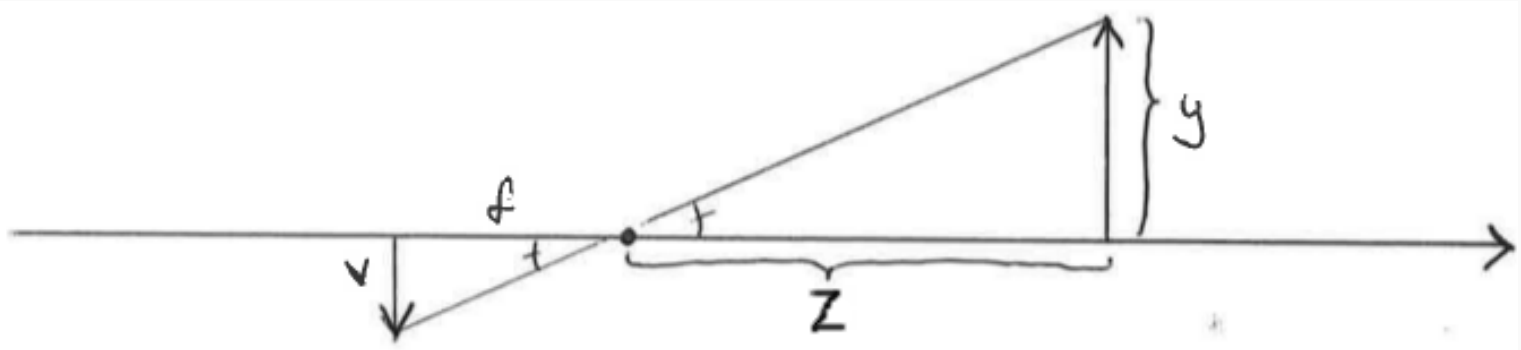
Assumptions

- Only direct lighting
- Indirect lighting can be simulated by ambient light
- All surfaces are perfectly diffuse
    - look the same from each angle
- All lights are points

Camera

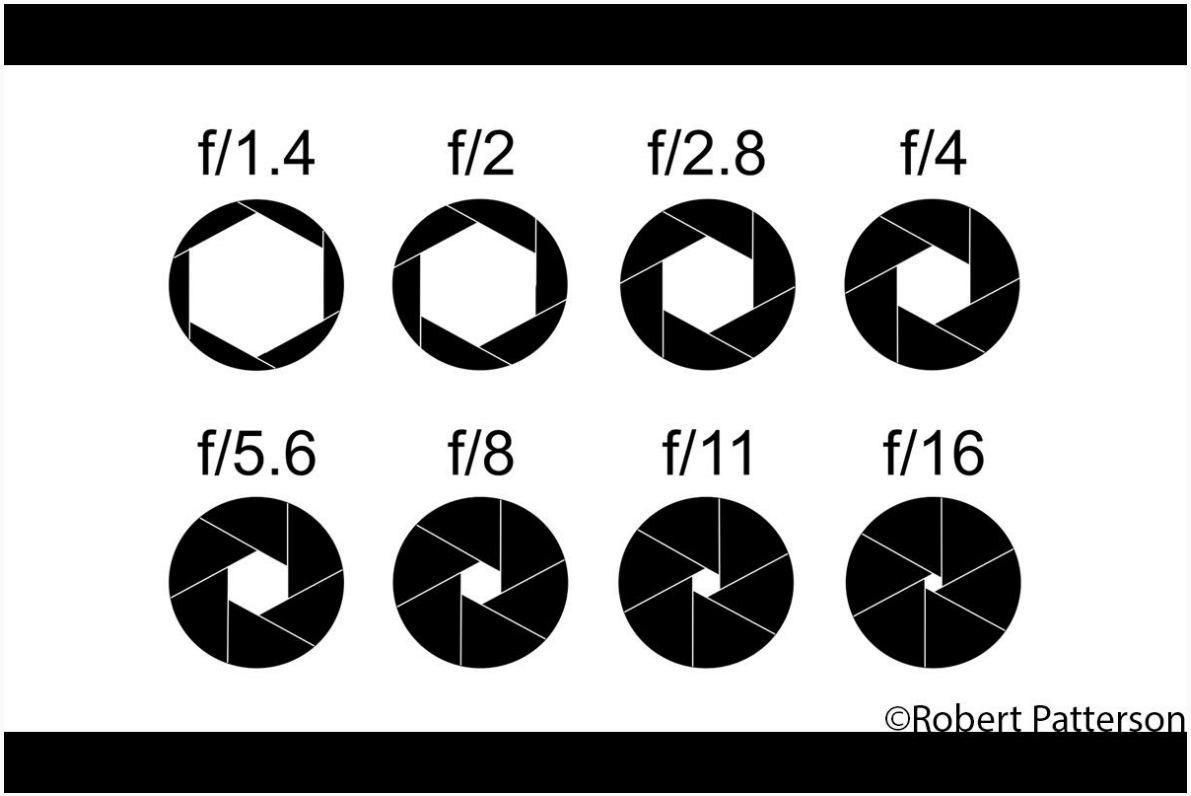
- Cameras create images
- Lens
  - captures rays
- Focal length
- Apature

Pinhole camera



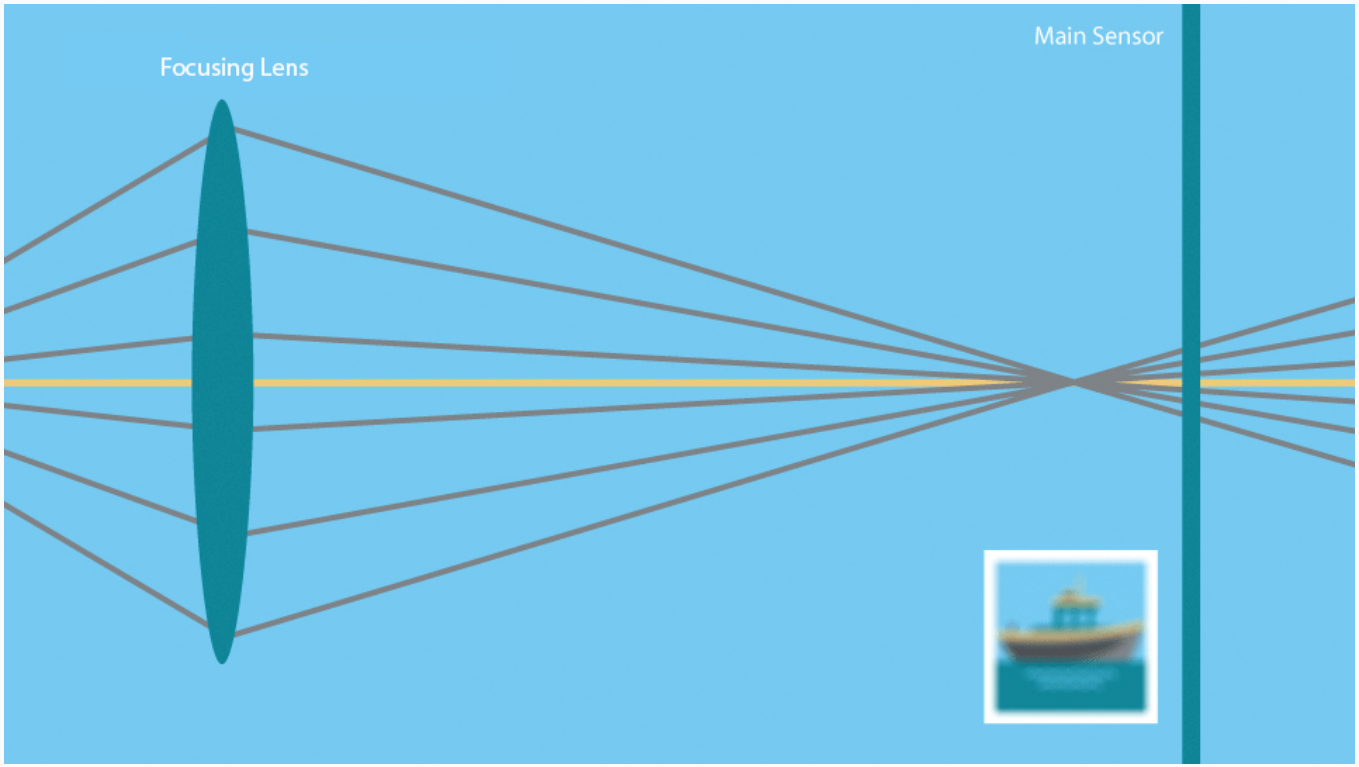
- Simplest form camera, where one ray of light lights up each pixel
- Defined by a *location*, *focal length*, *view-direction* and *up-direction*
- is this a realistic camera?

Apature[6]



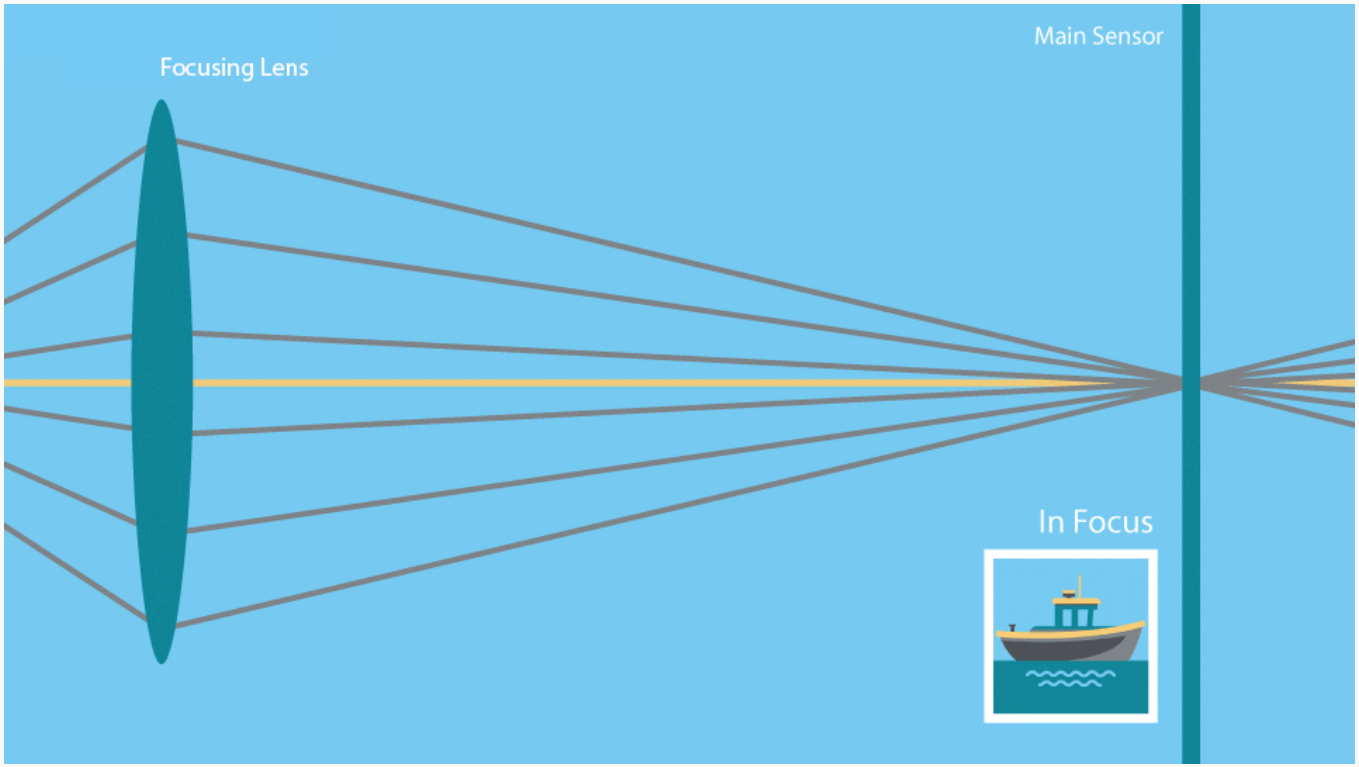
[6] https://geneseodigitalphotography.wordpress.com/learn-about-your-camera/reference-sheets/aperture/

Depth-of-Field[7]



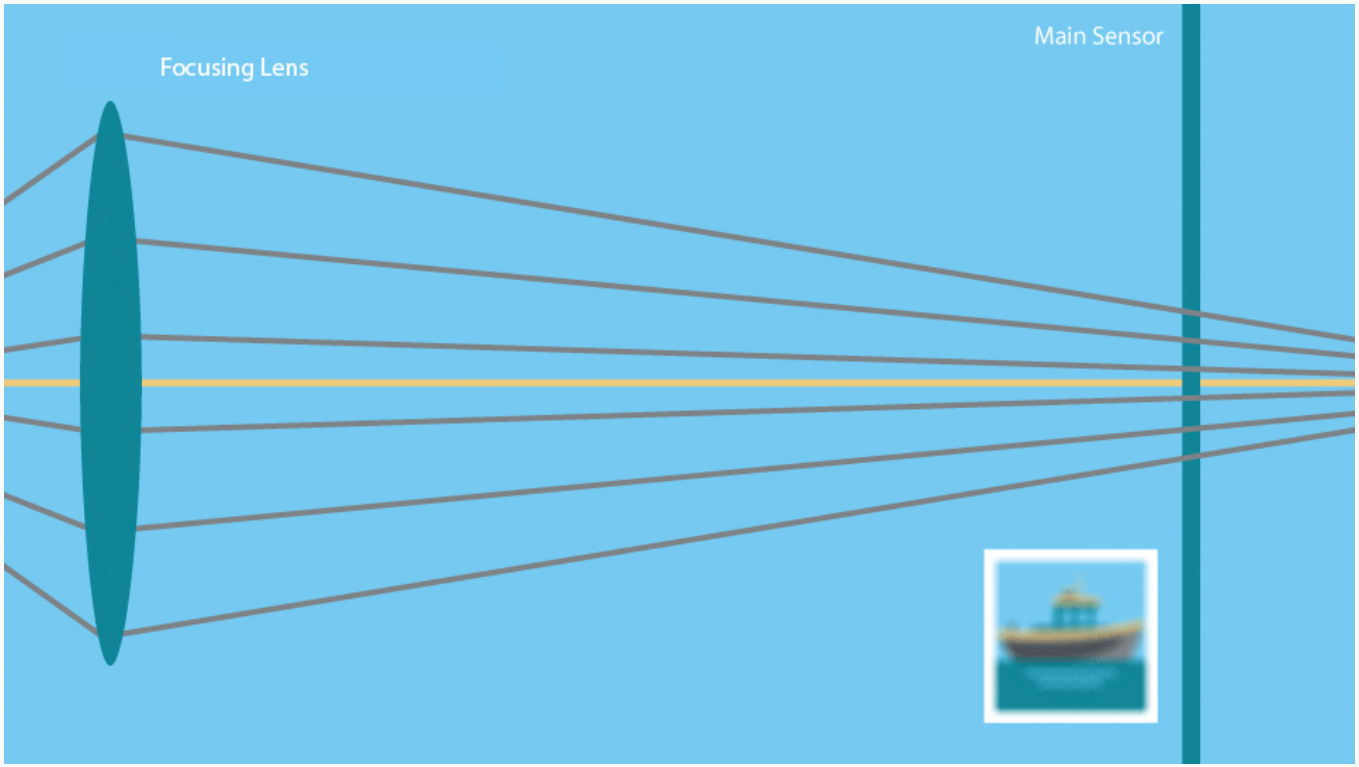
[7] https://www.bhphotovideo.com/explora/photography/tips-and-solutions/how-focus-works

Depth-of-Field[7]



[7] https://www.bhphotovideo.com/explora/photography/tips-and-solutions/how-focus-works

Depth-of-Field[7]



[7] https://www.bhphotovideo.com/explora/photography/tips-and-solutions/how-focus-works

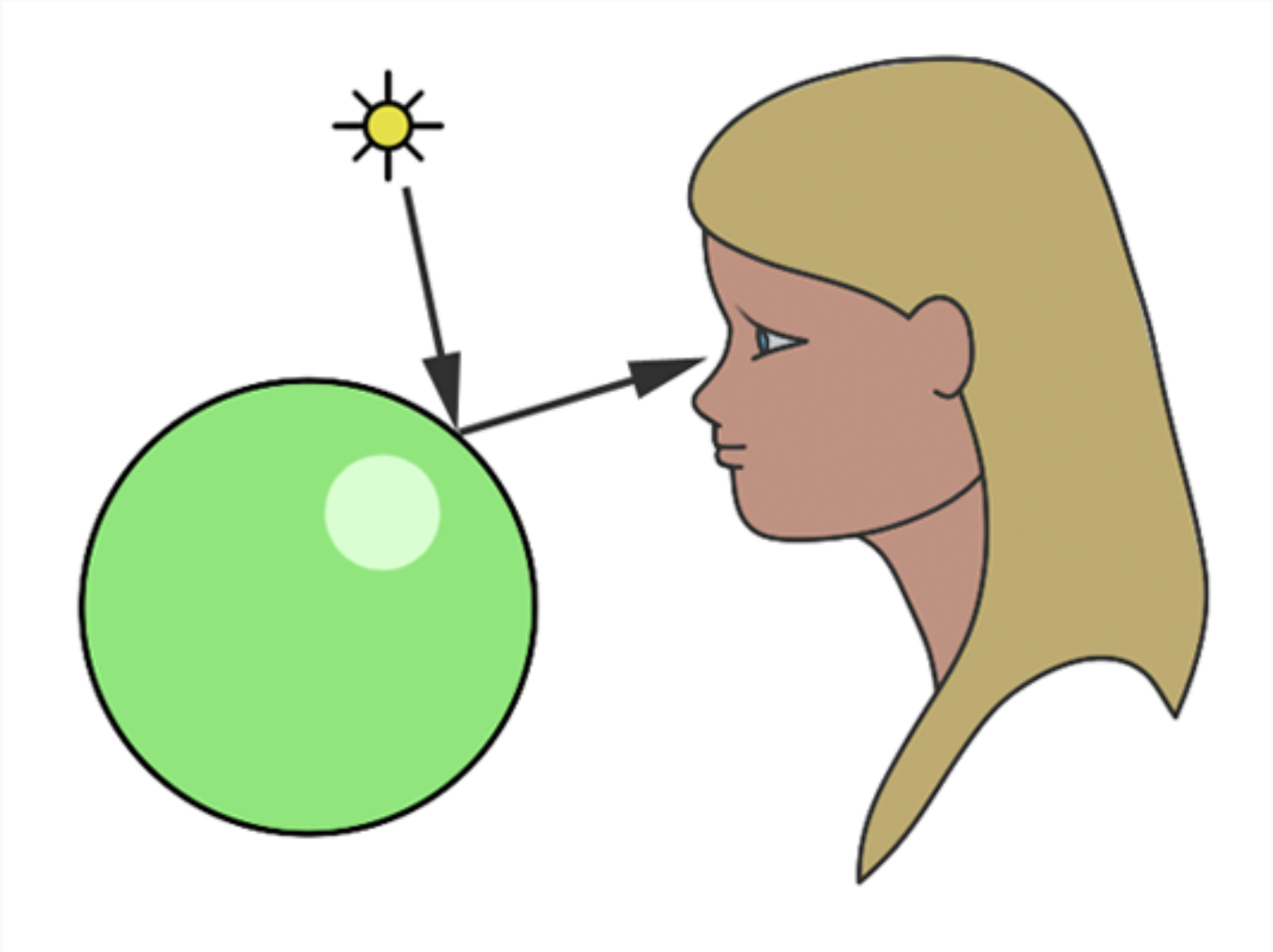
Lens Flare

Assumptions

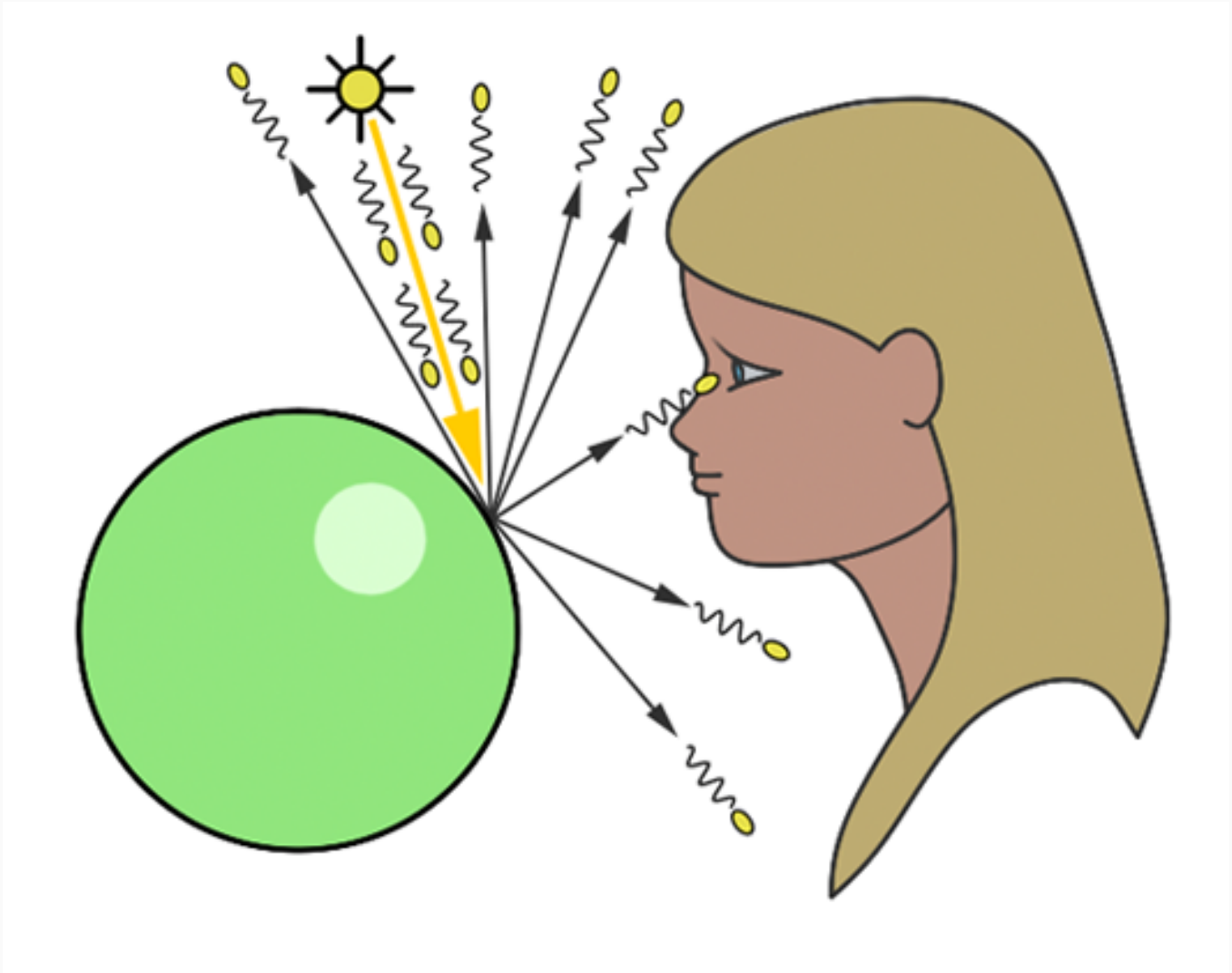
- Pinhole camera
  - everything is in focus
  - point aperture
- No lens
  - no lens flare
  - no lens distortion

Raytracing

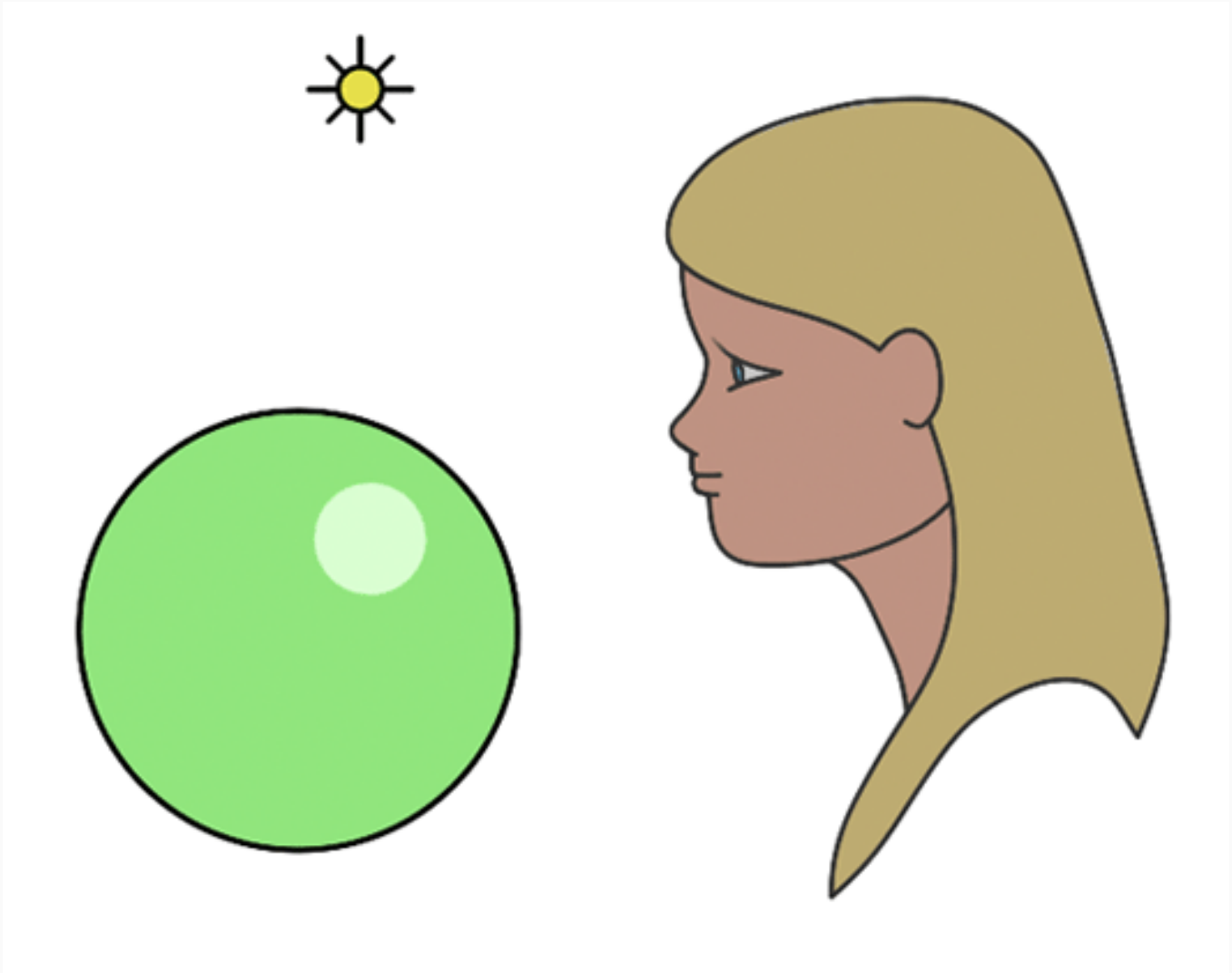
Raytracing



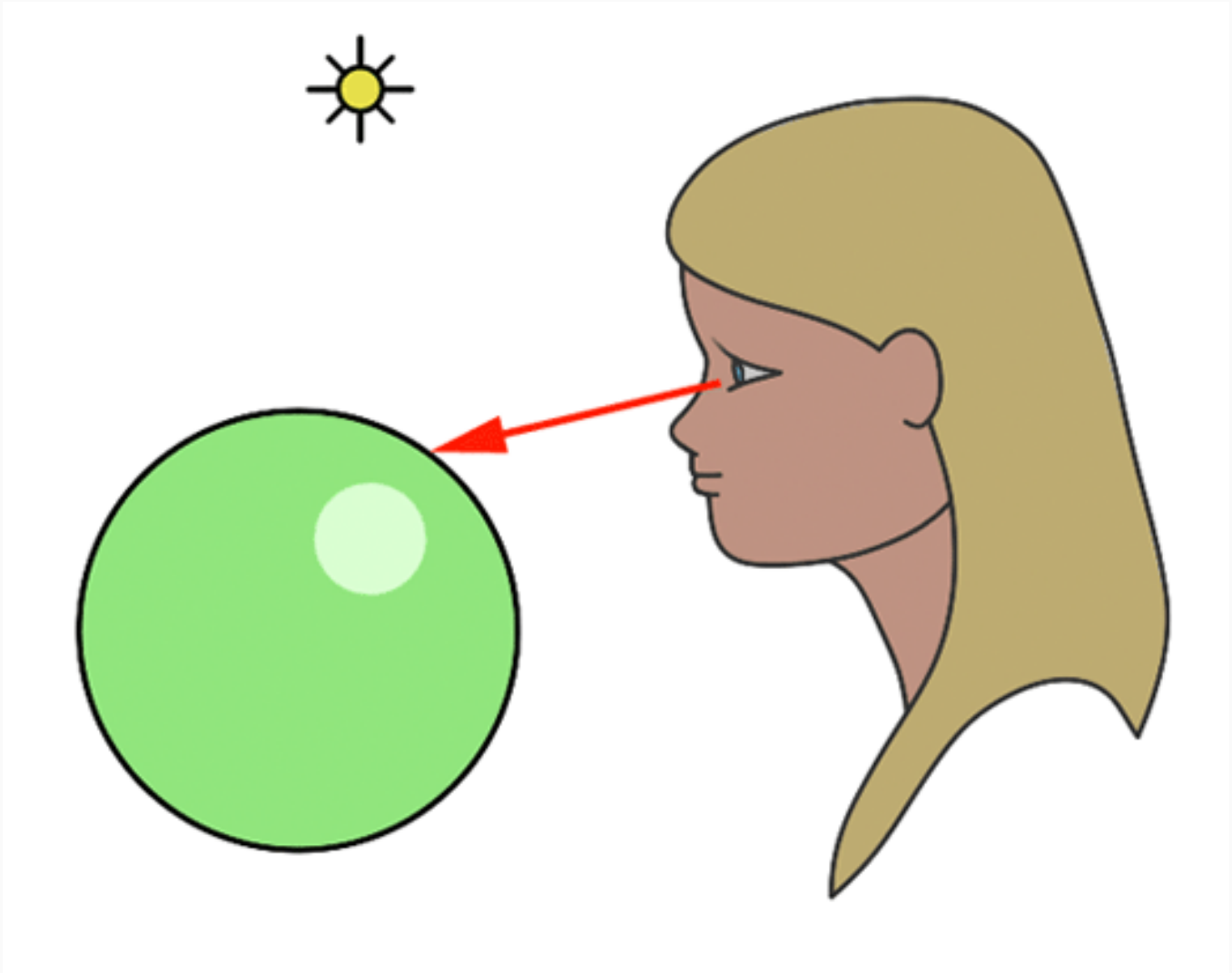
Raytracing



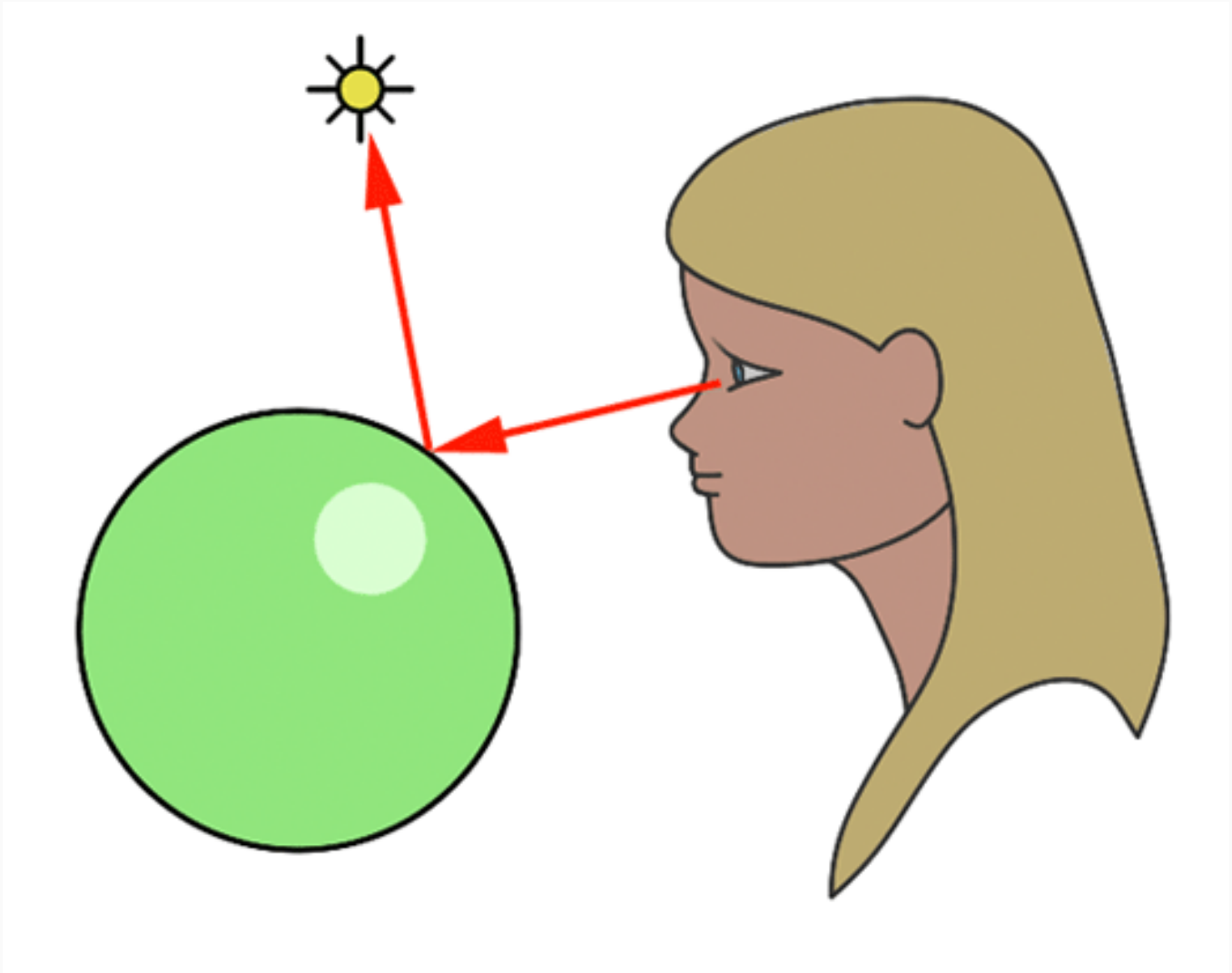
Raytracing



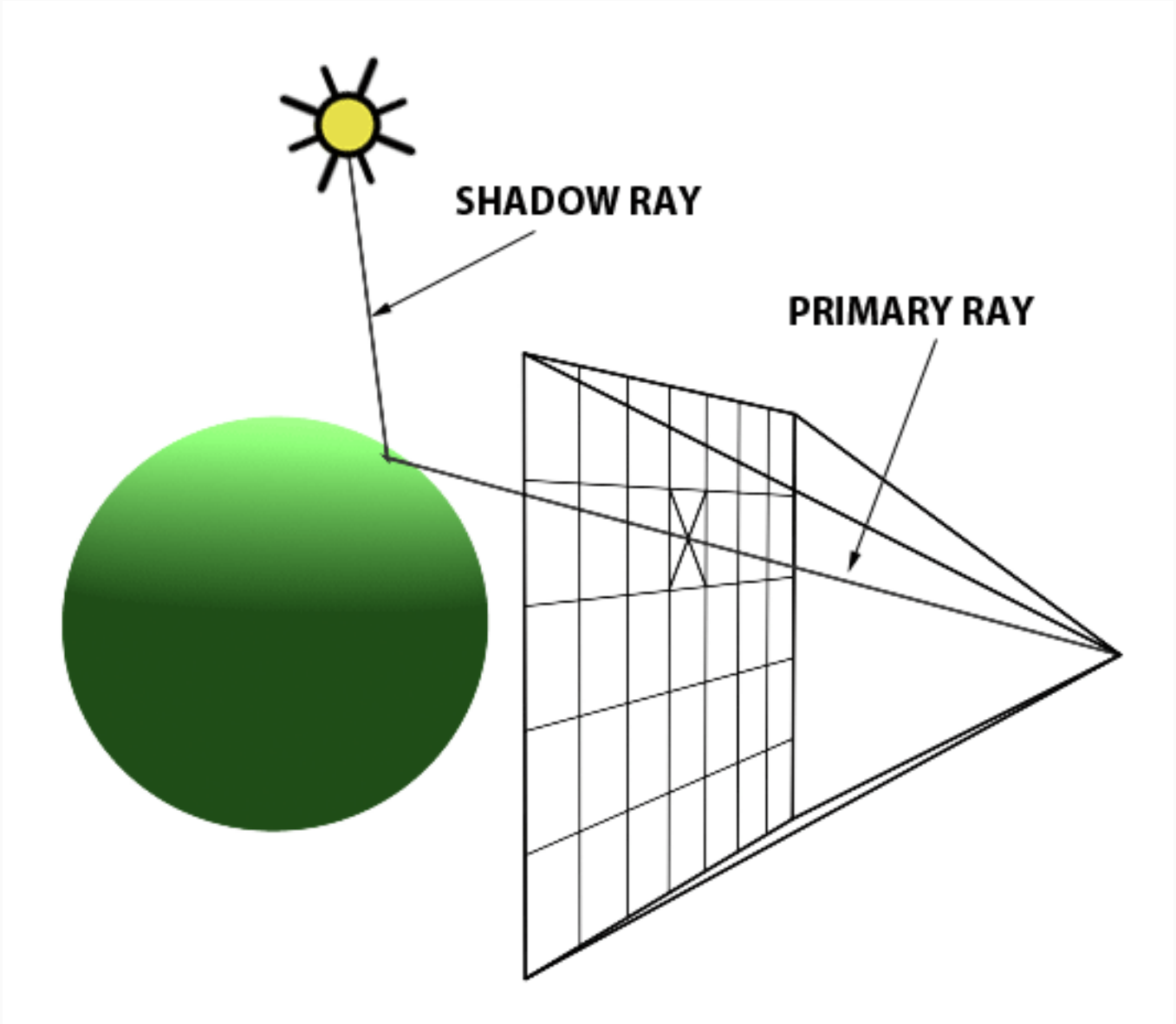
Raytracing



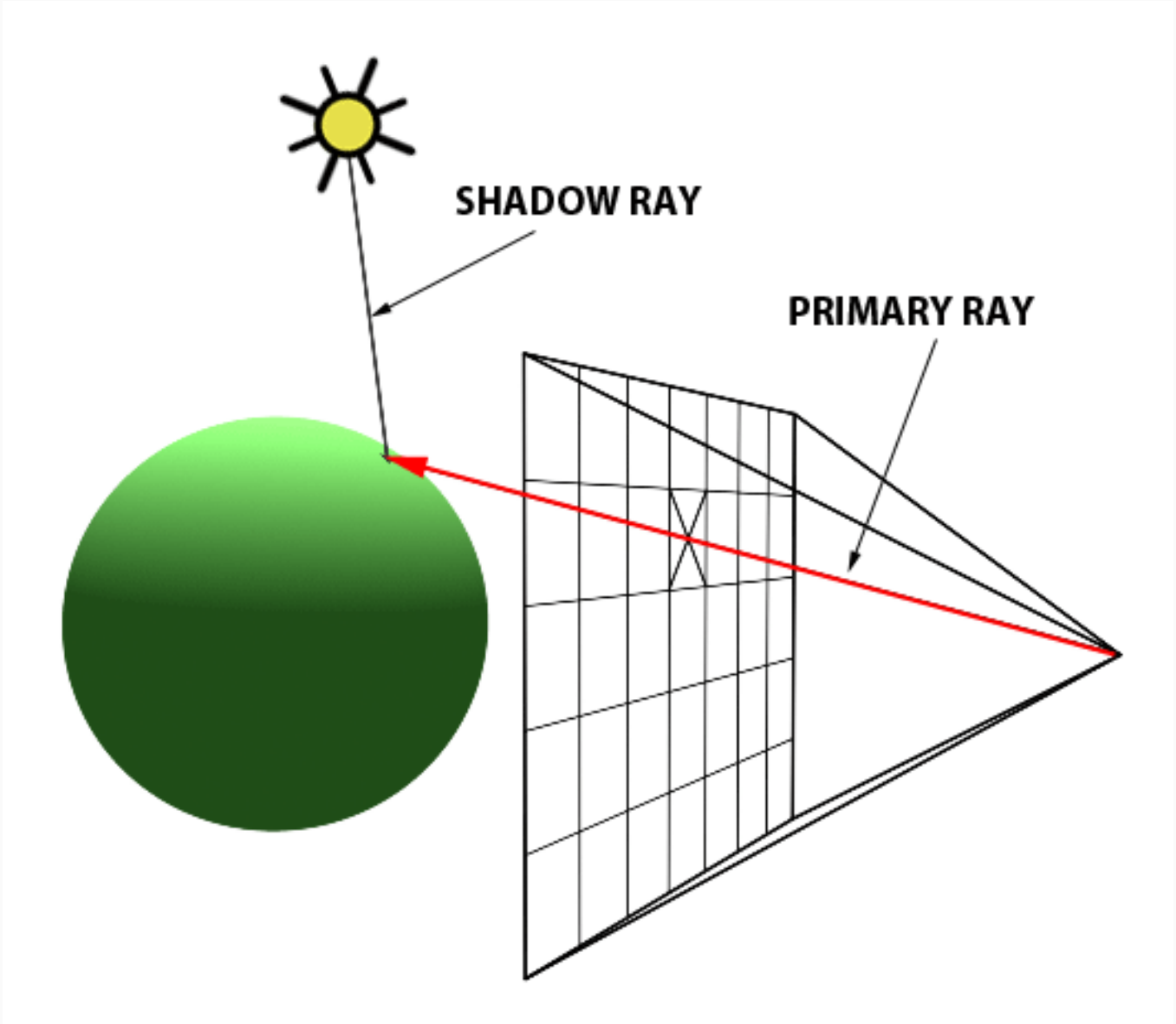
Raytracing



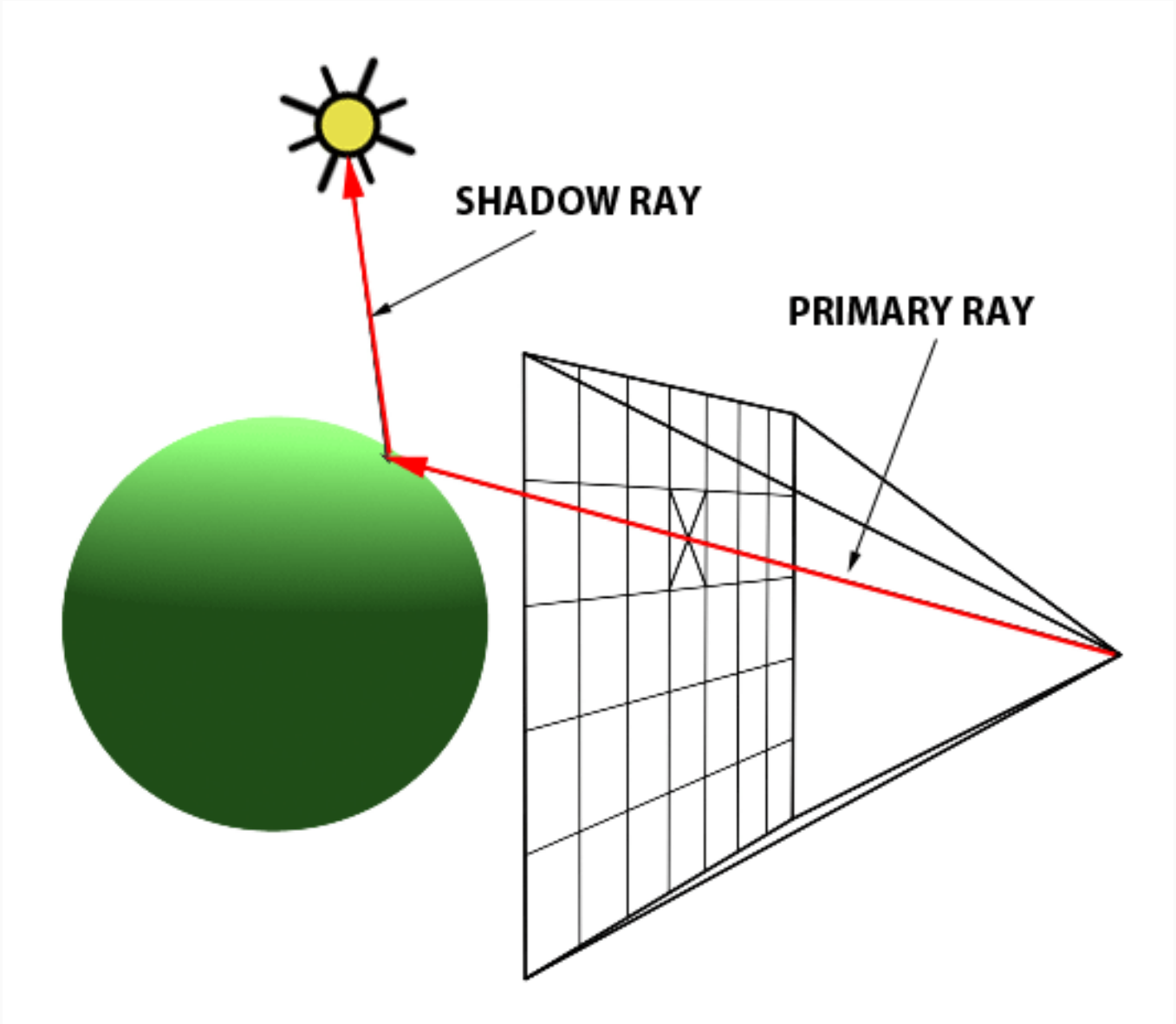
Raytracing



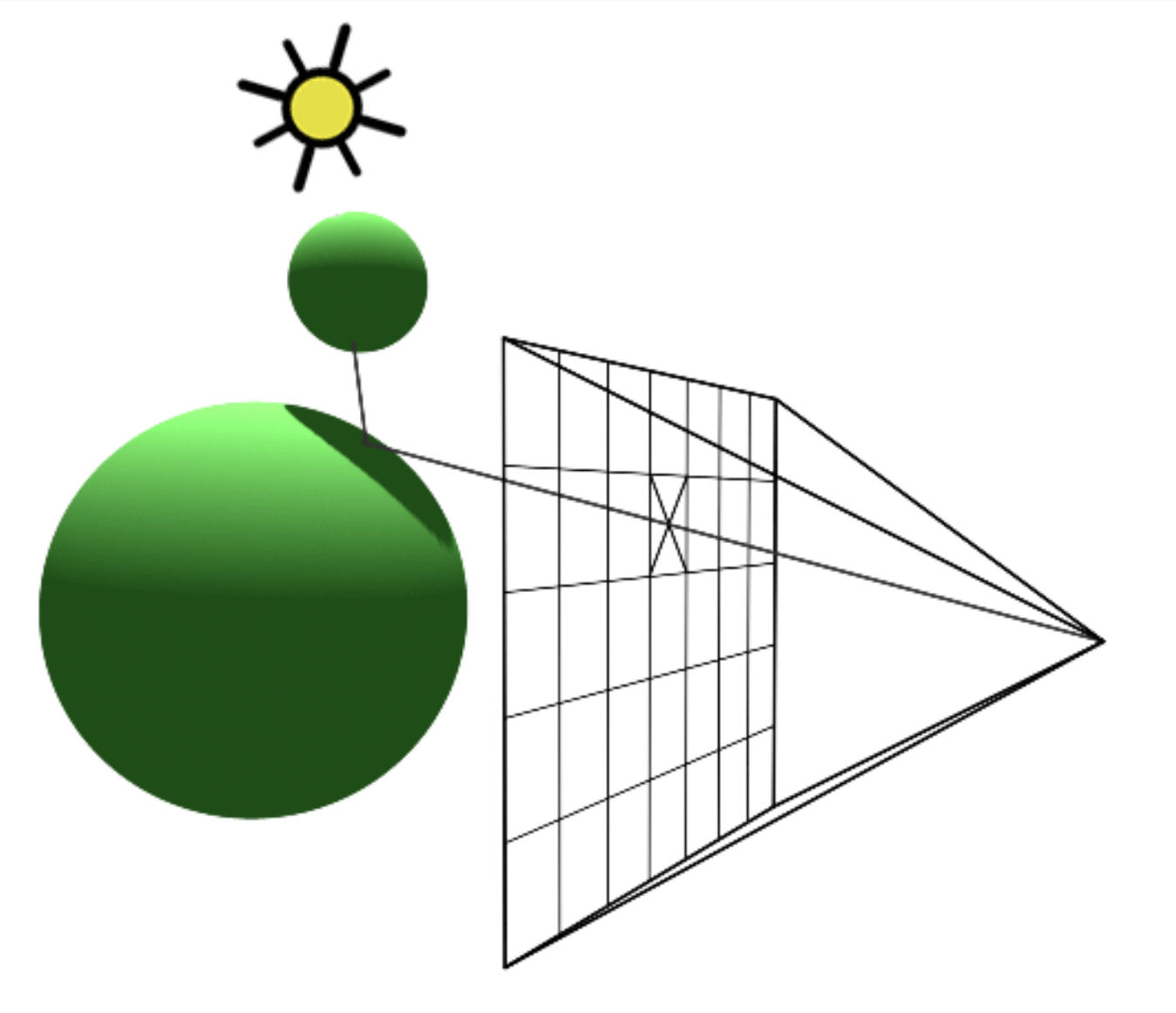
Raytracing



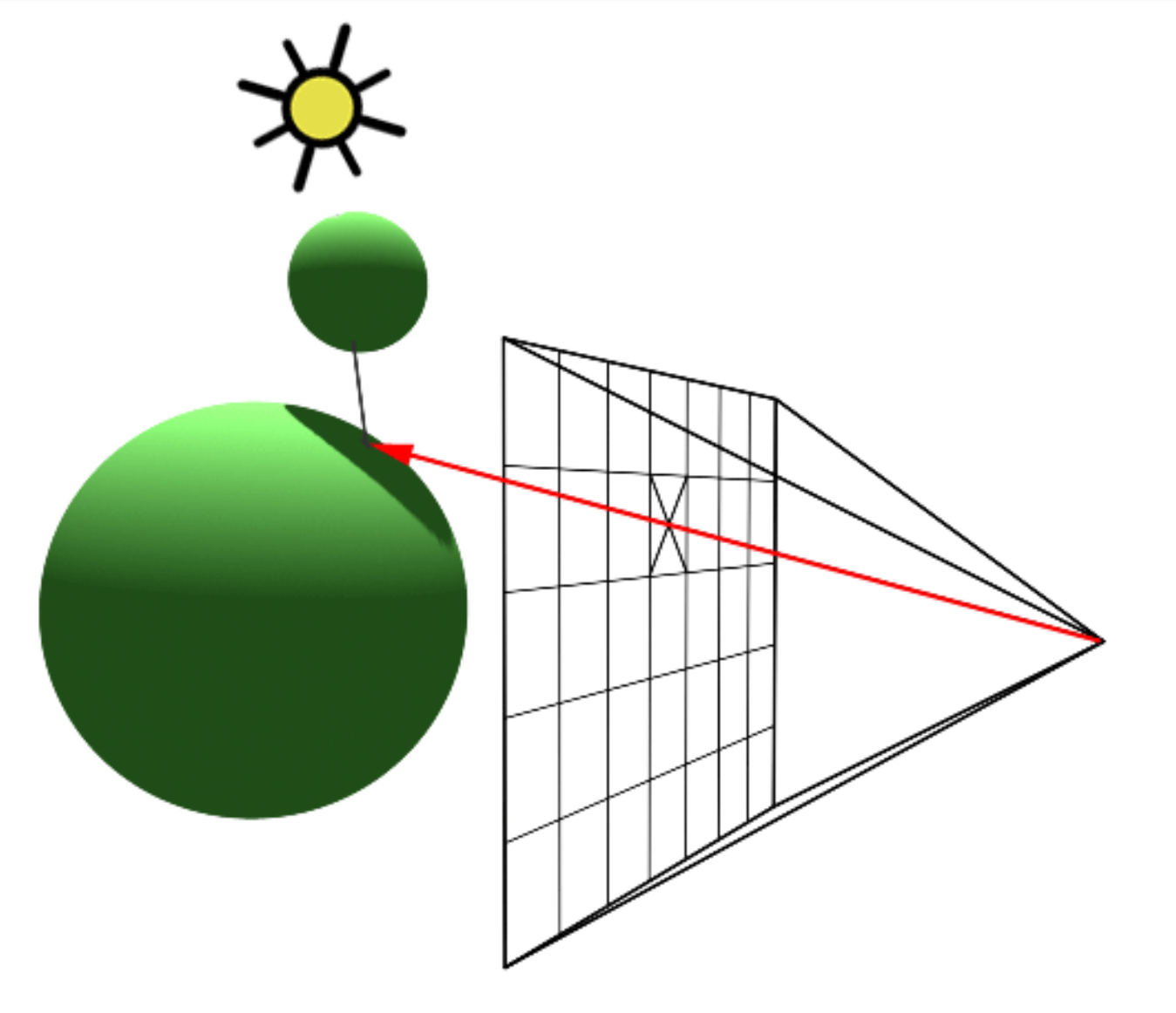
Raytracing



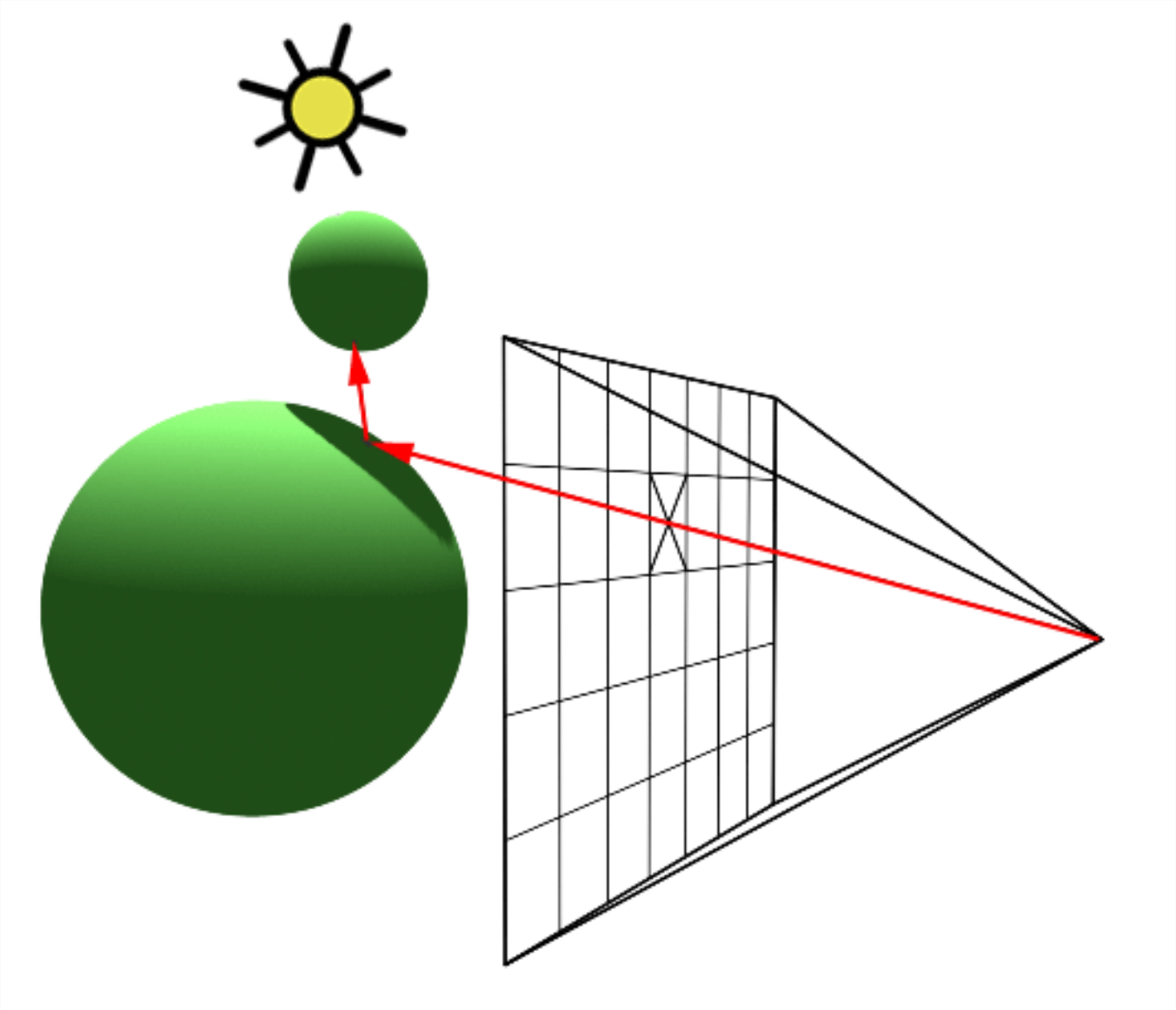
Raytracing



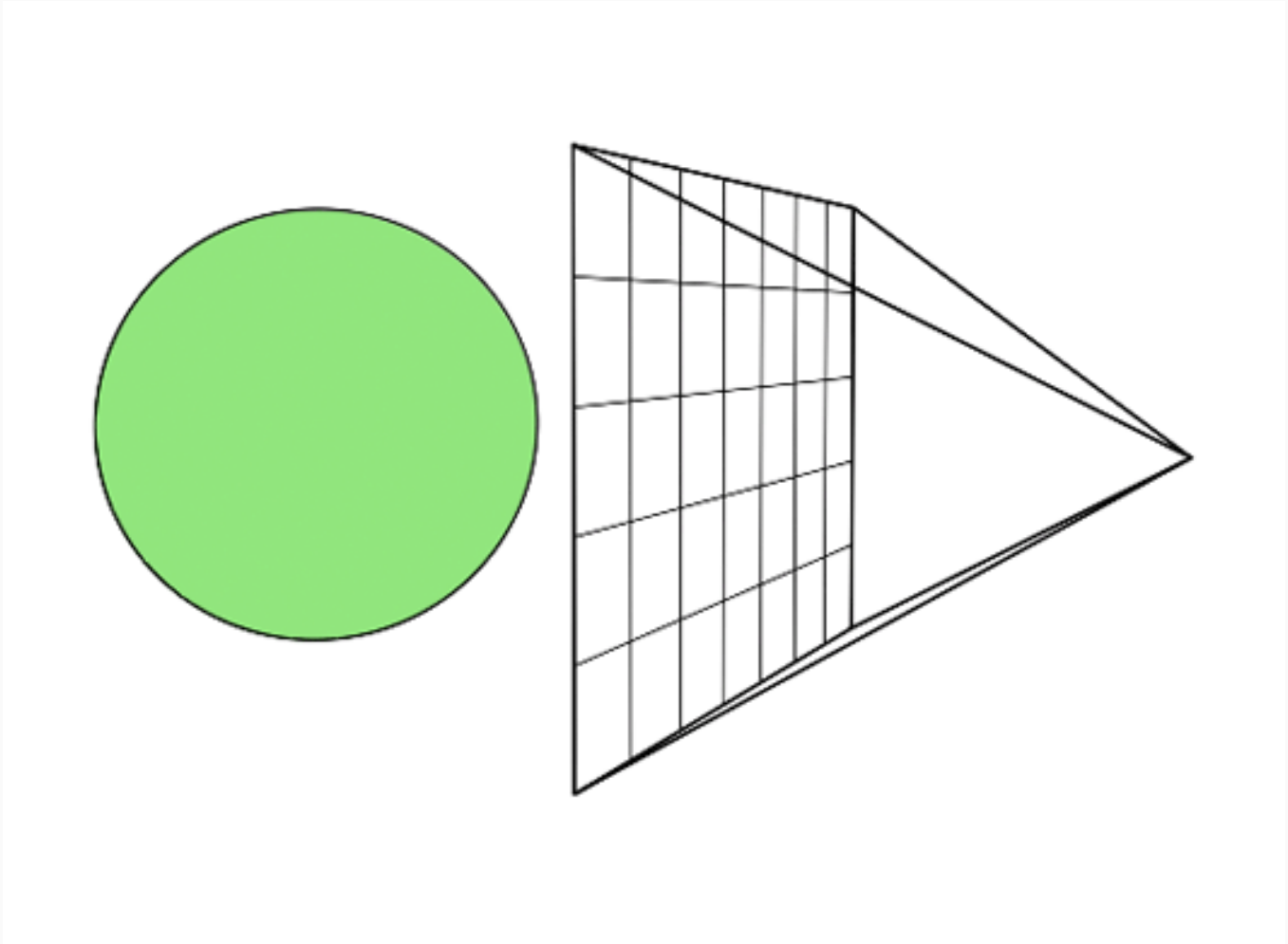
Raytracing



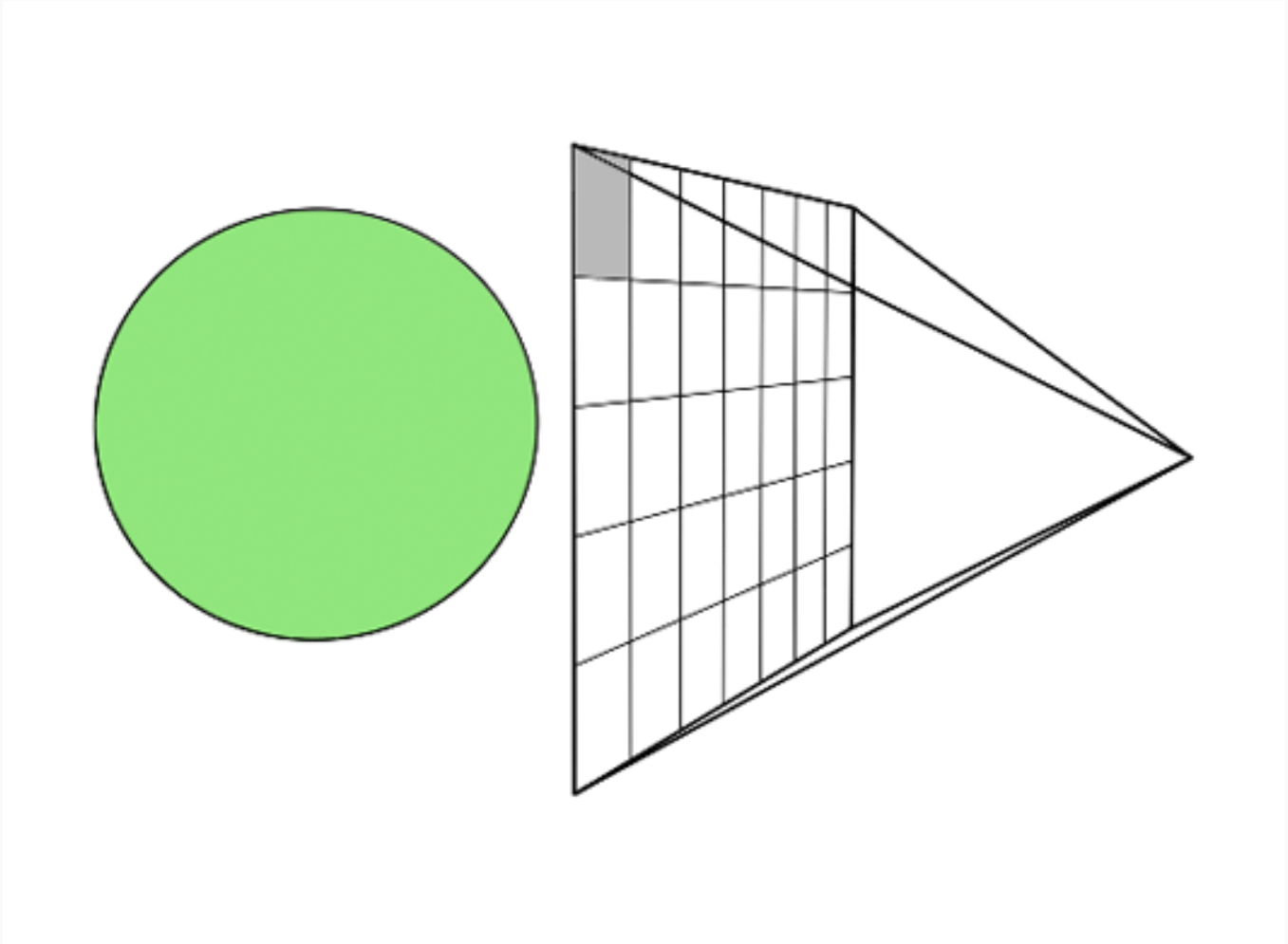
Raytracing



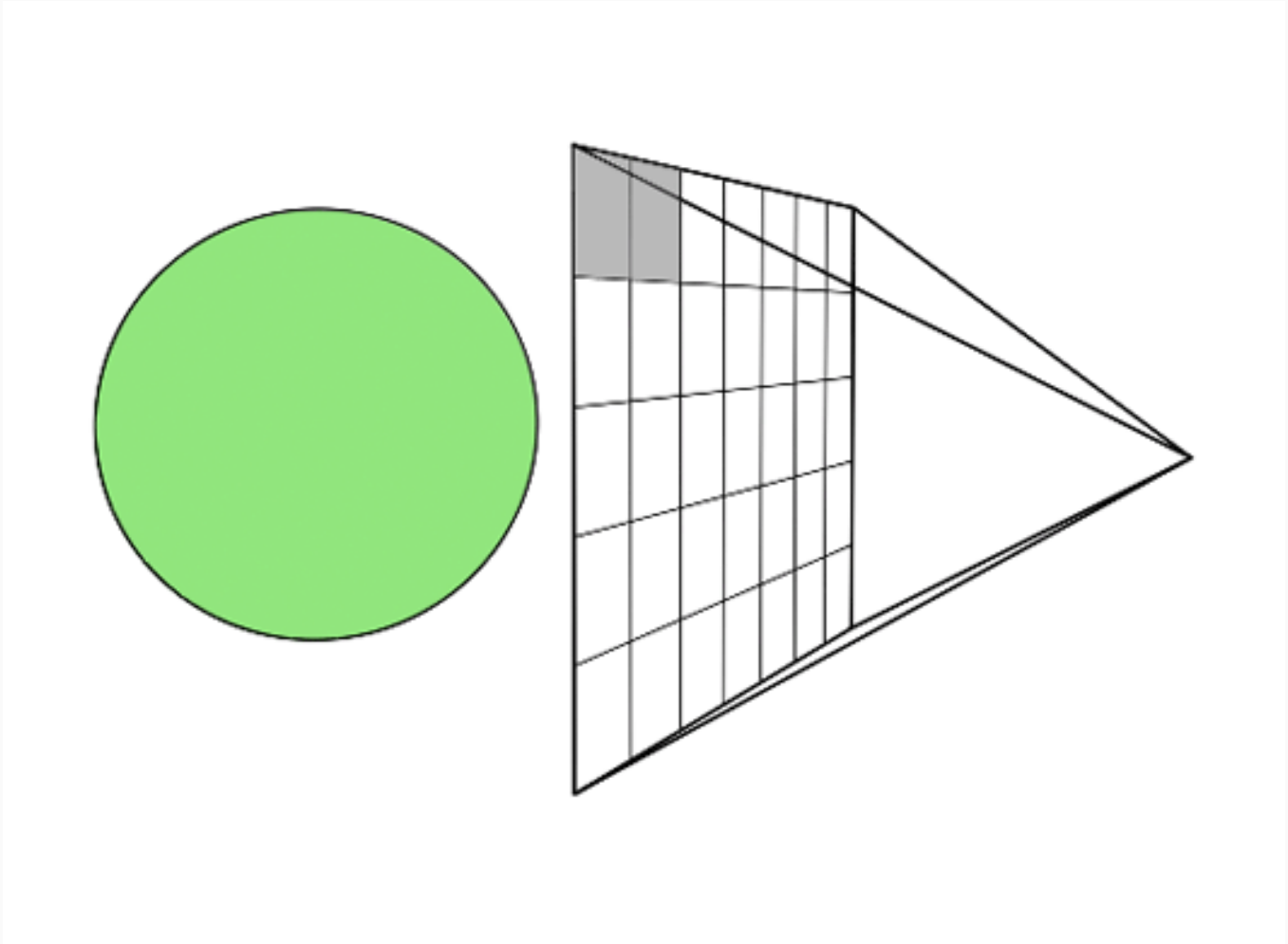
Raycasting Anim



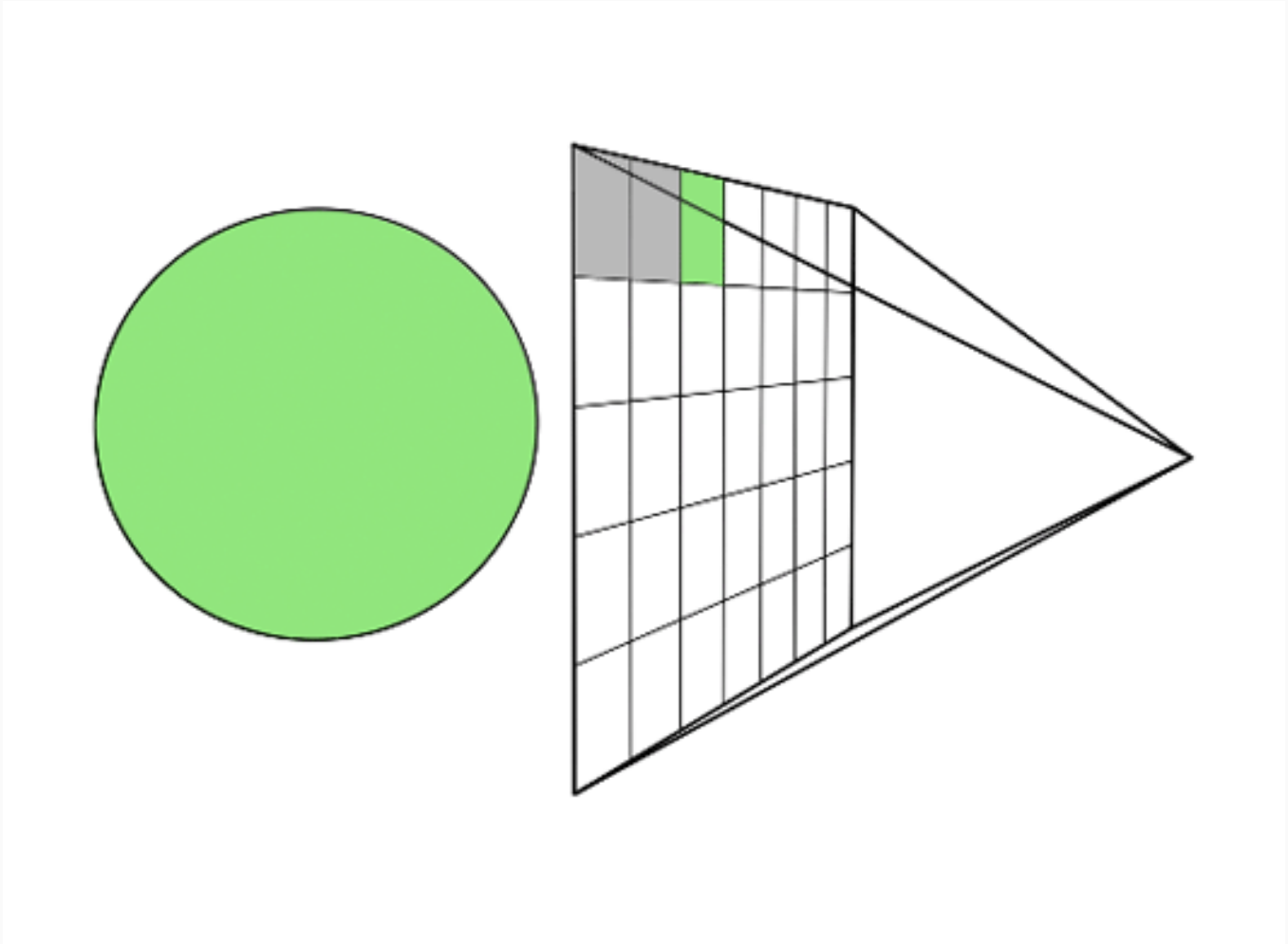
Raycasting Anim



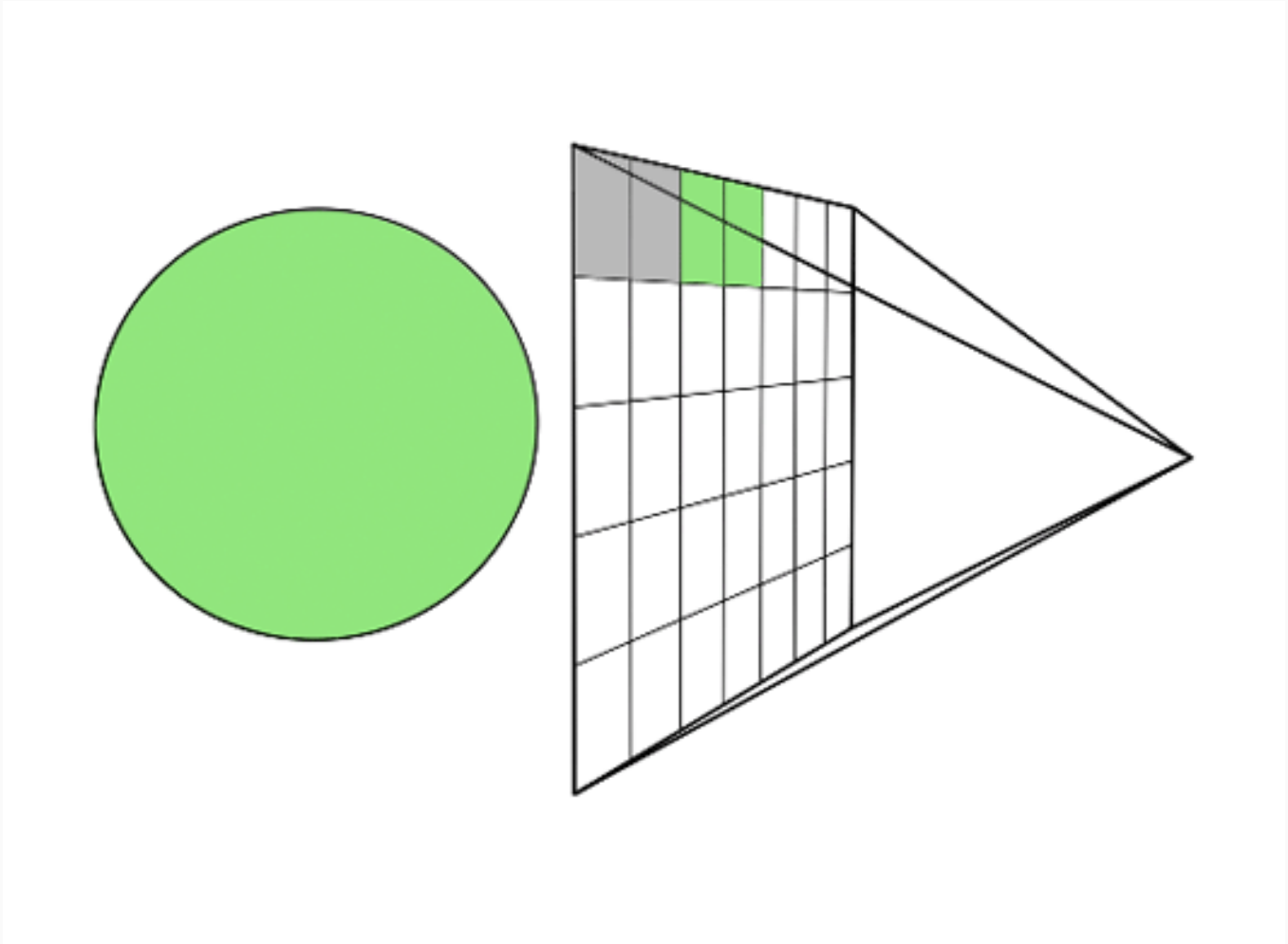
Raycasting Anim



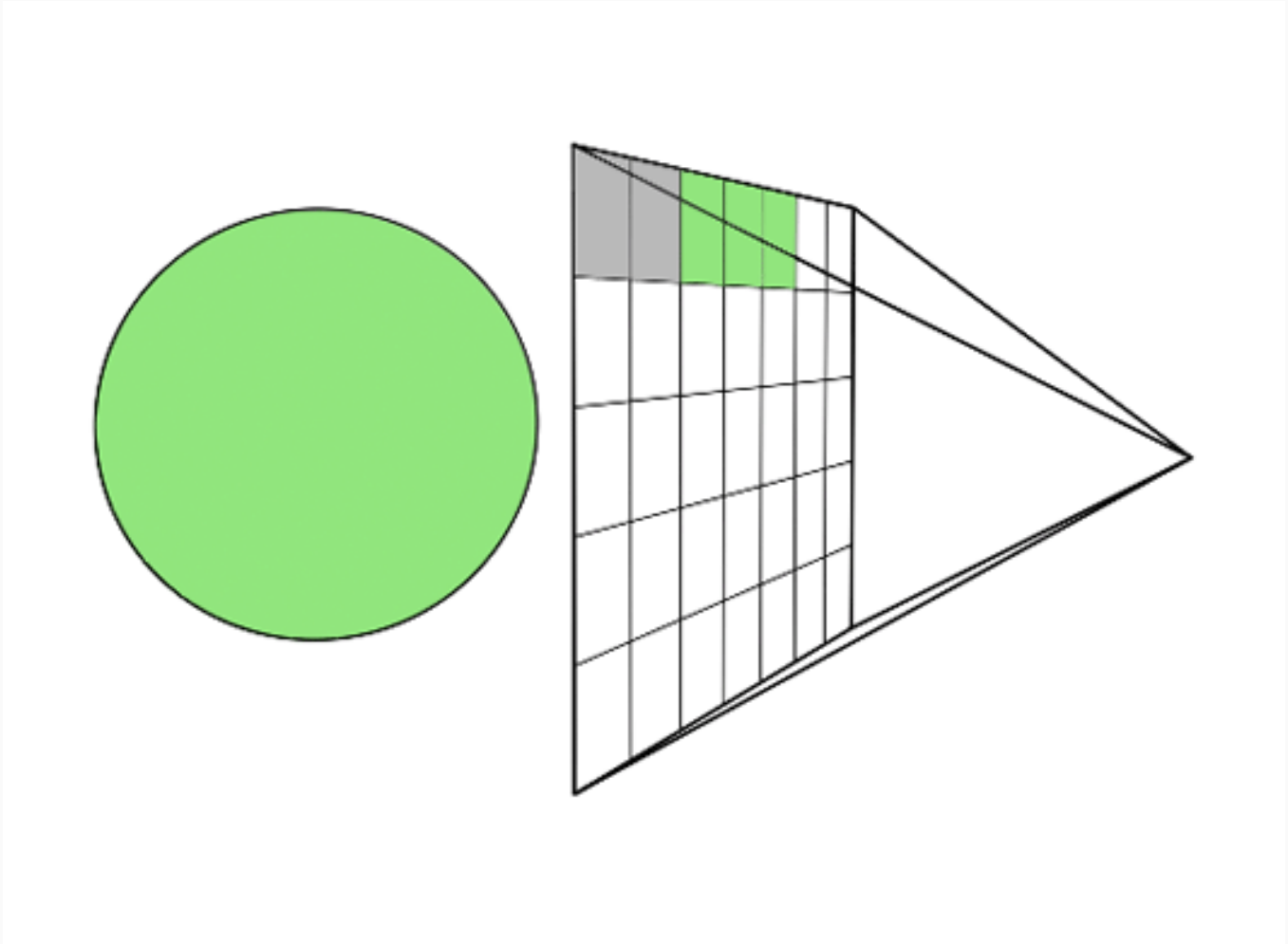
Raycasting Anim



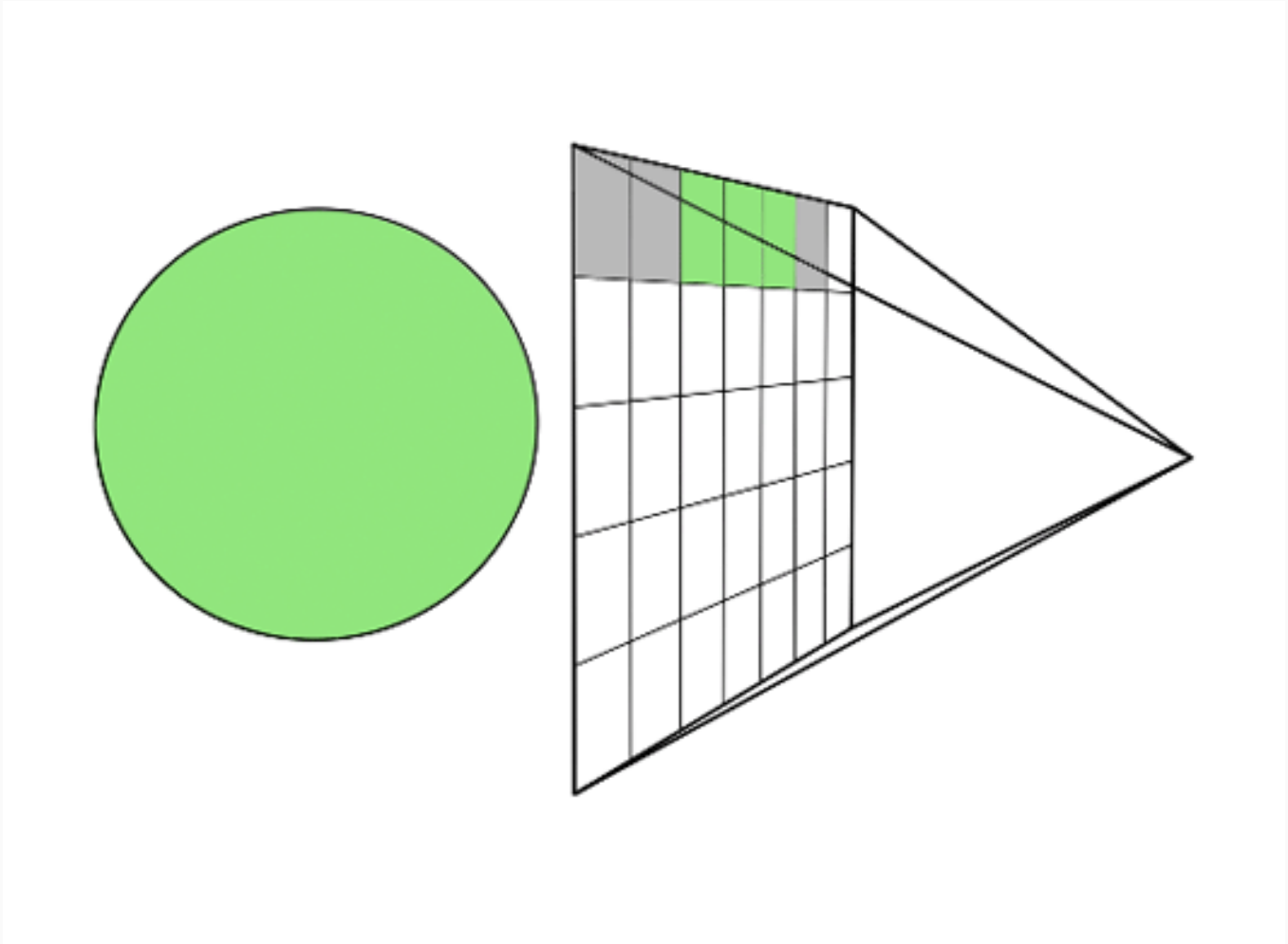
Raycasting Anim



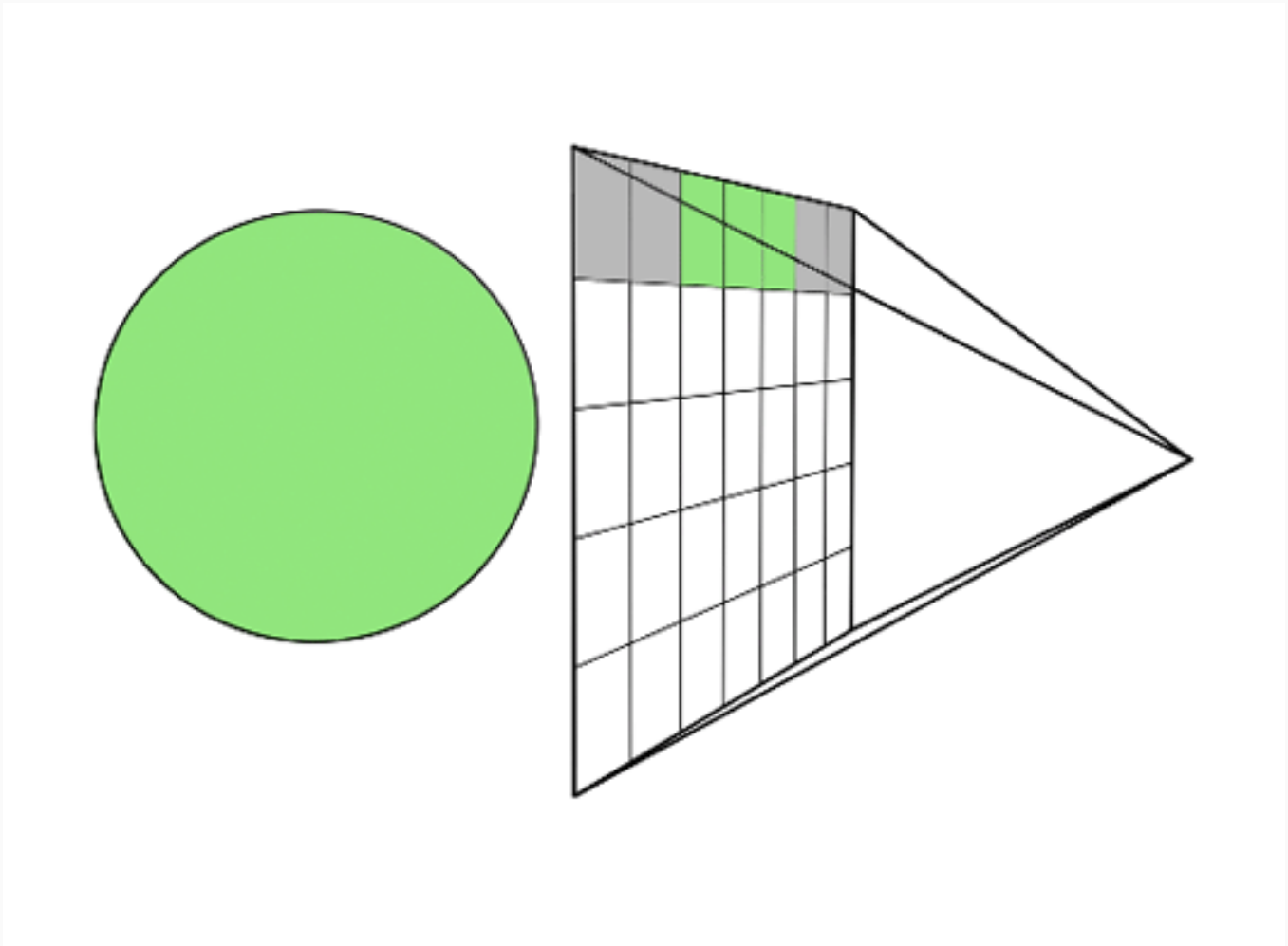
Raycasting Anim



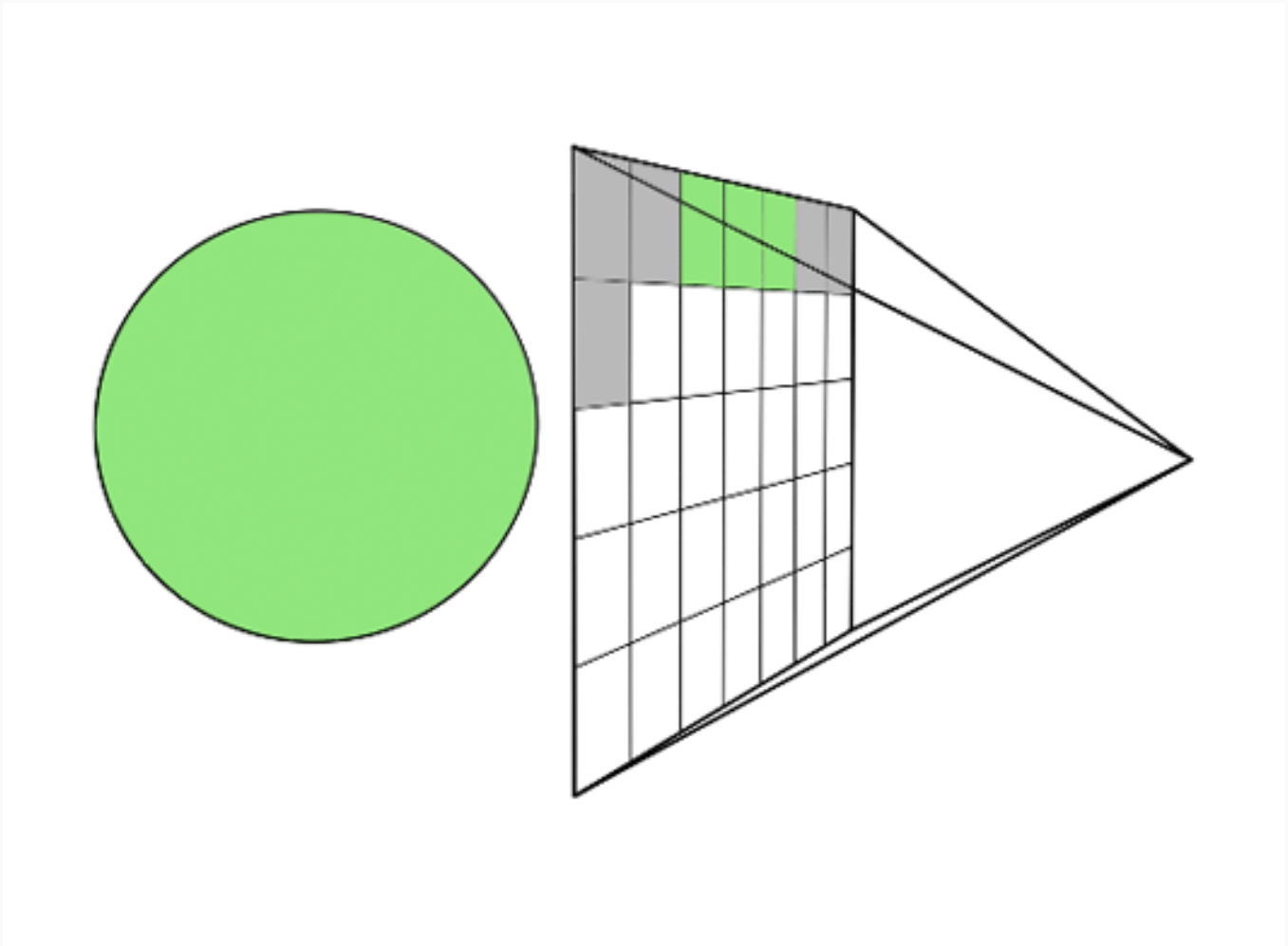
Raycasting Anim



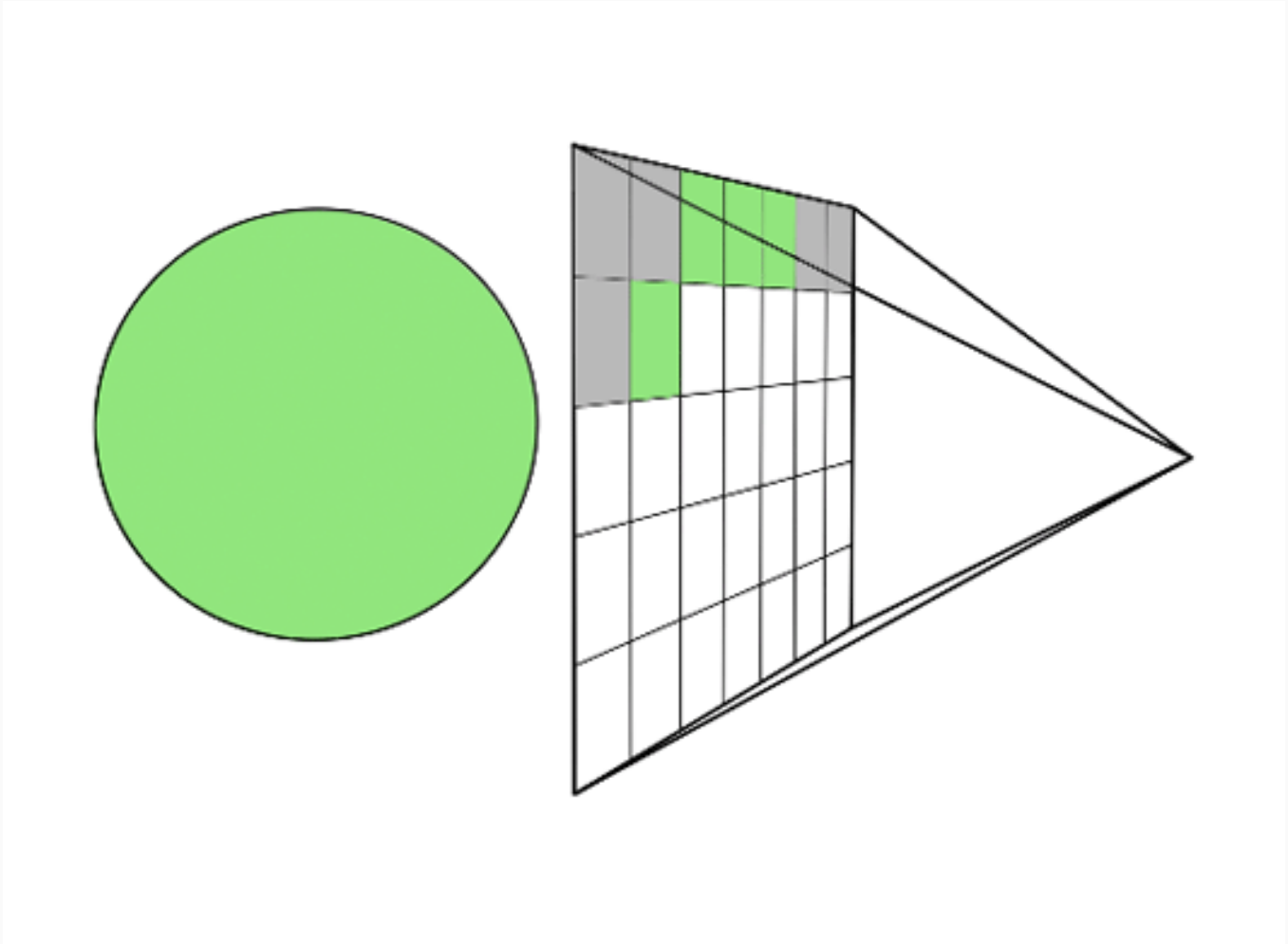
Raycasting Anim



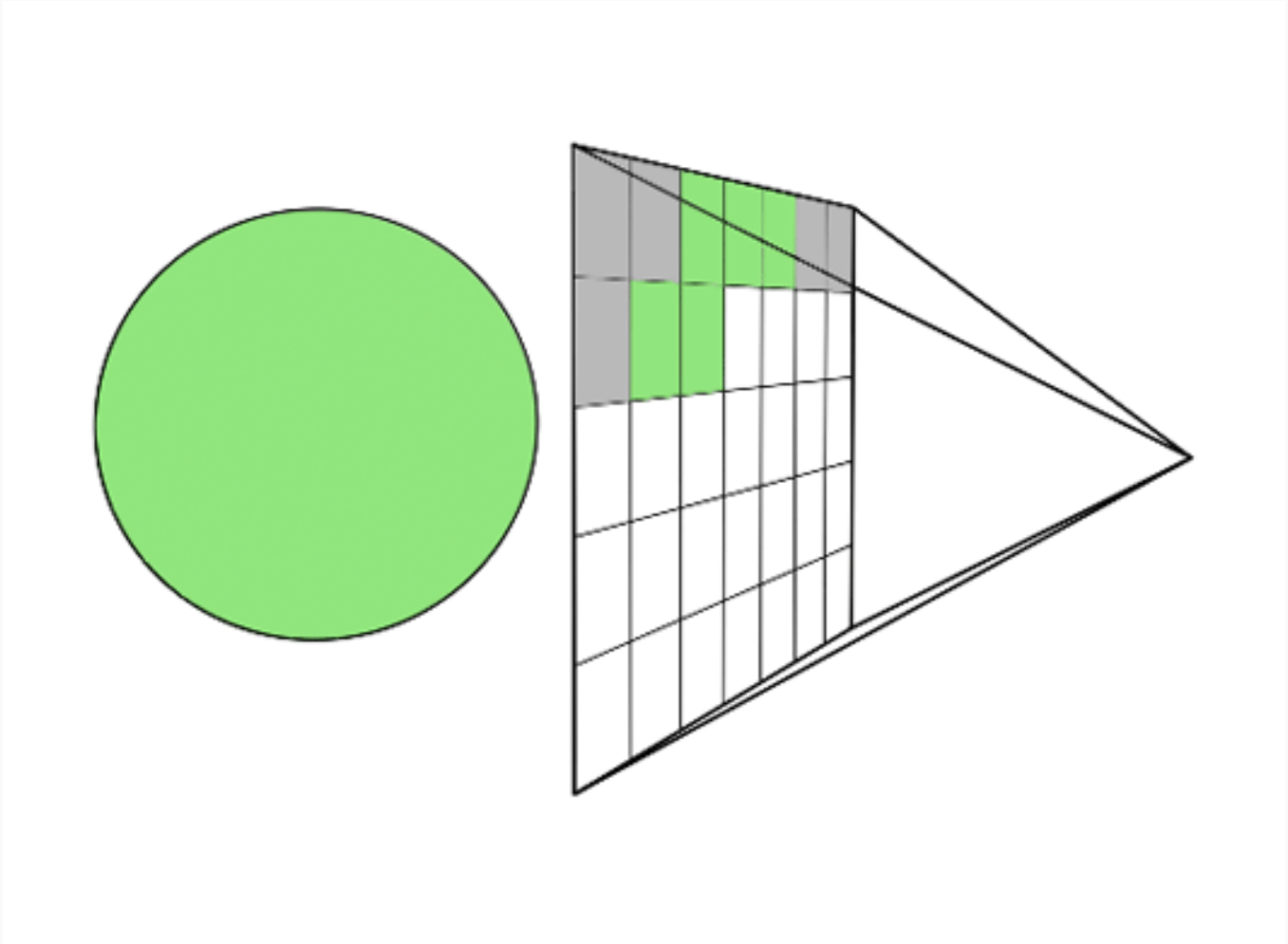
Raycasting Anim



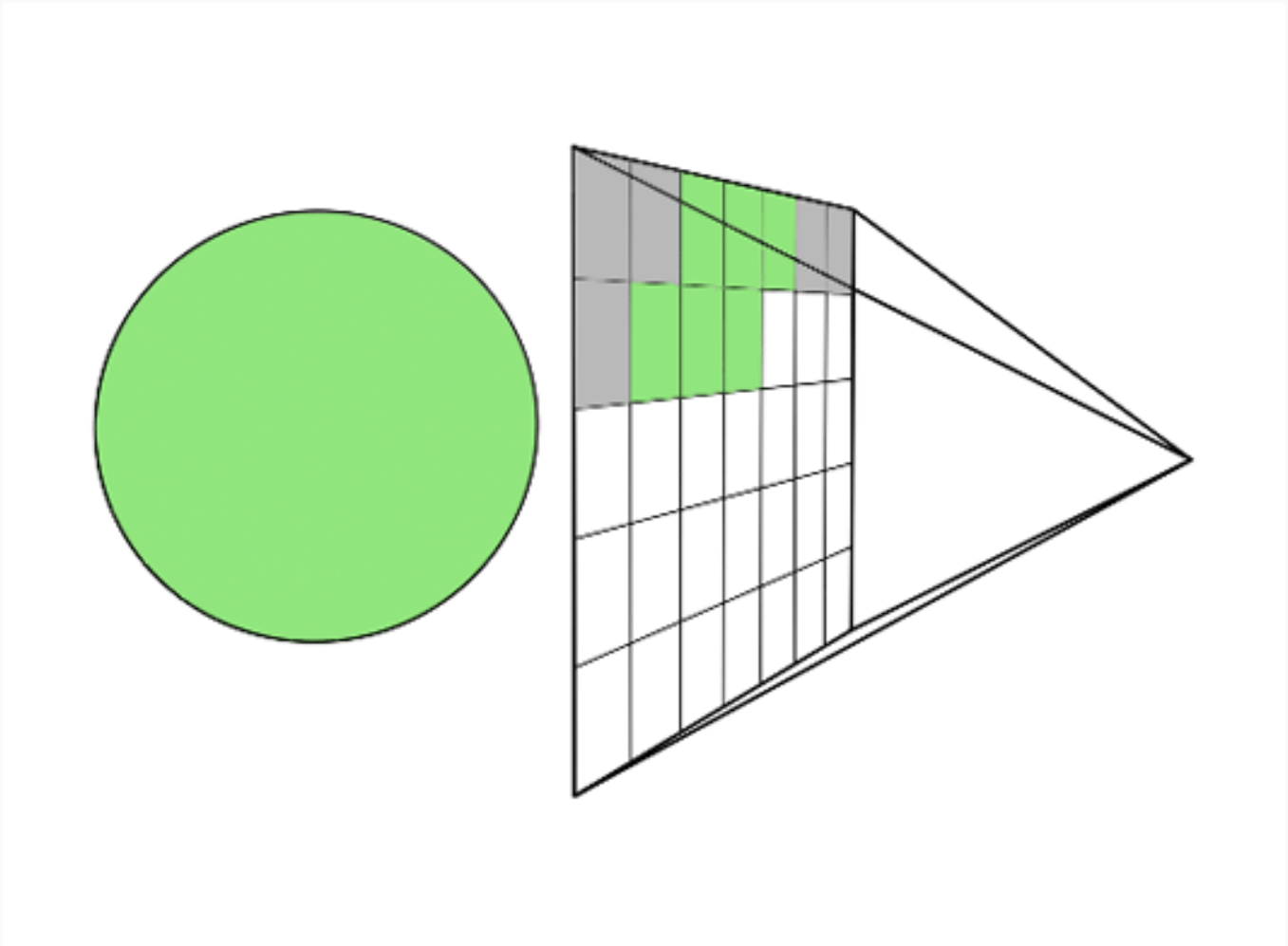
Raycasting Anim



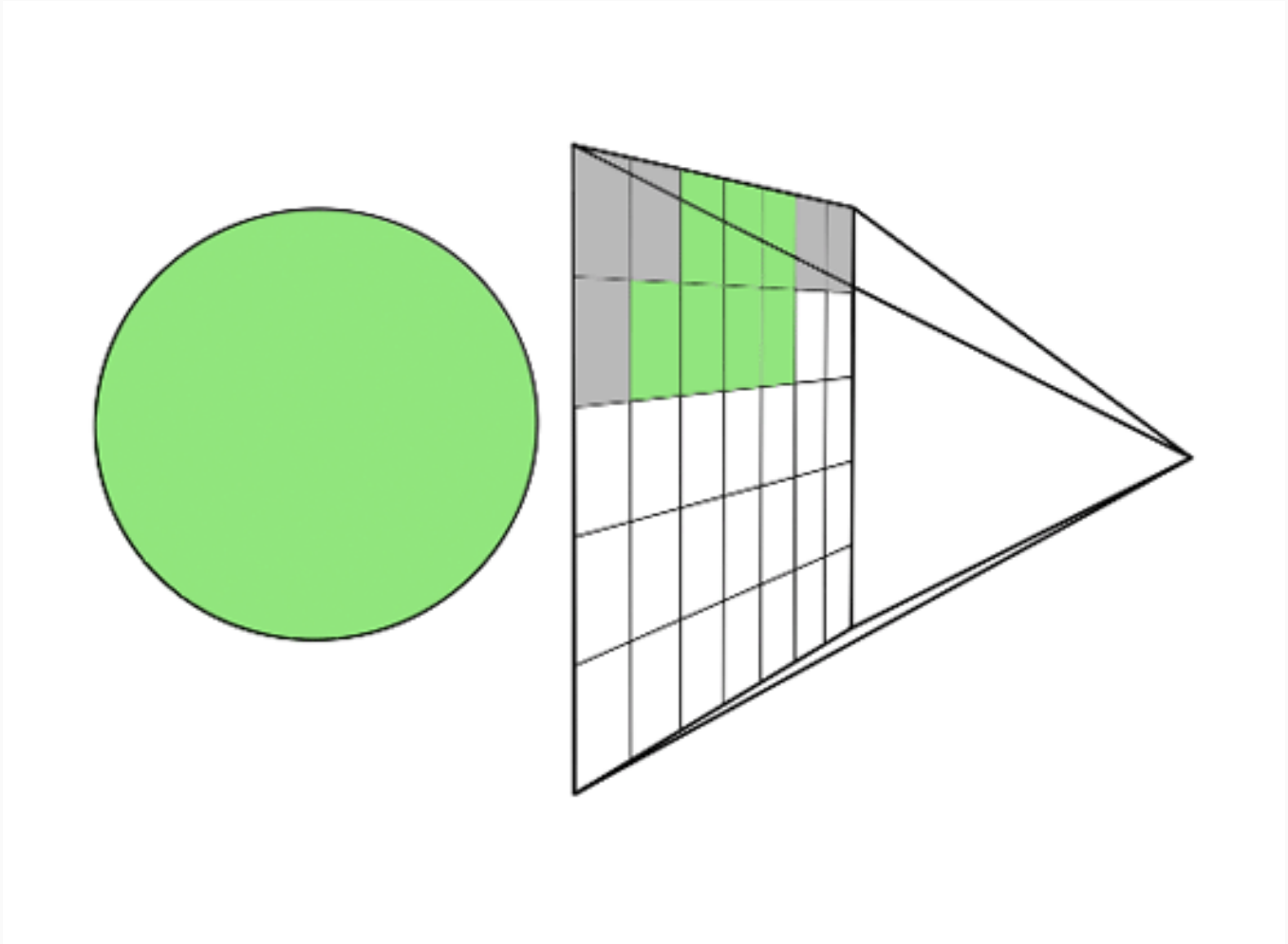
Raycasting Anim



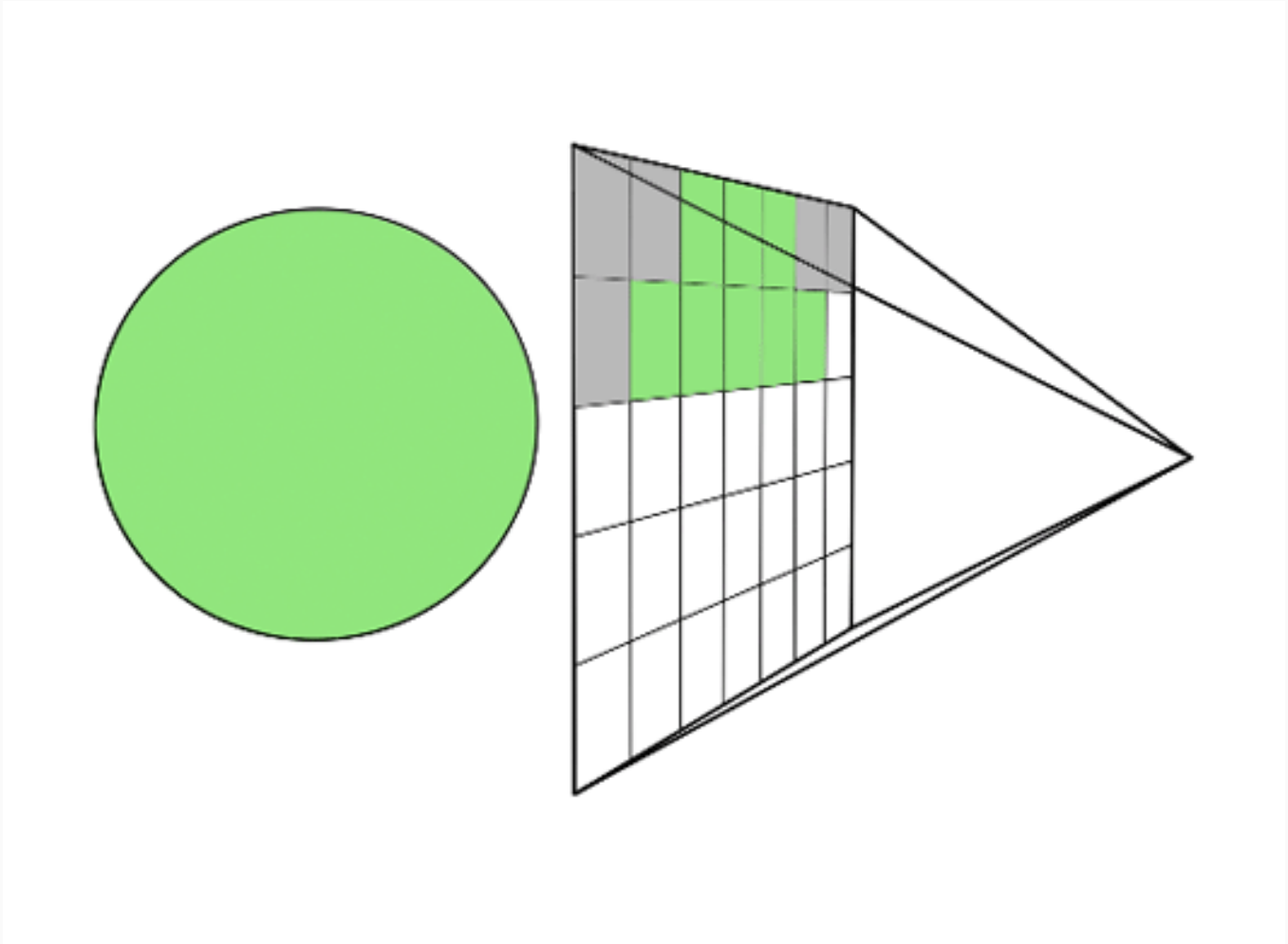
Raycasting Anim



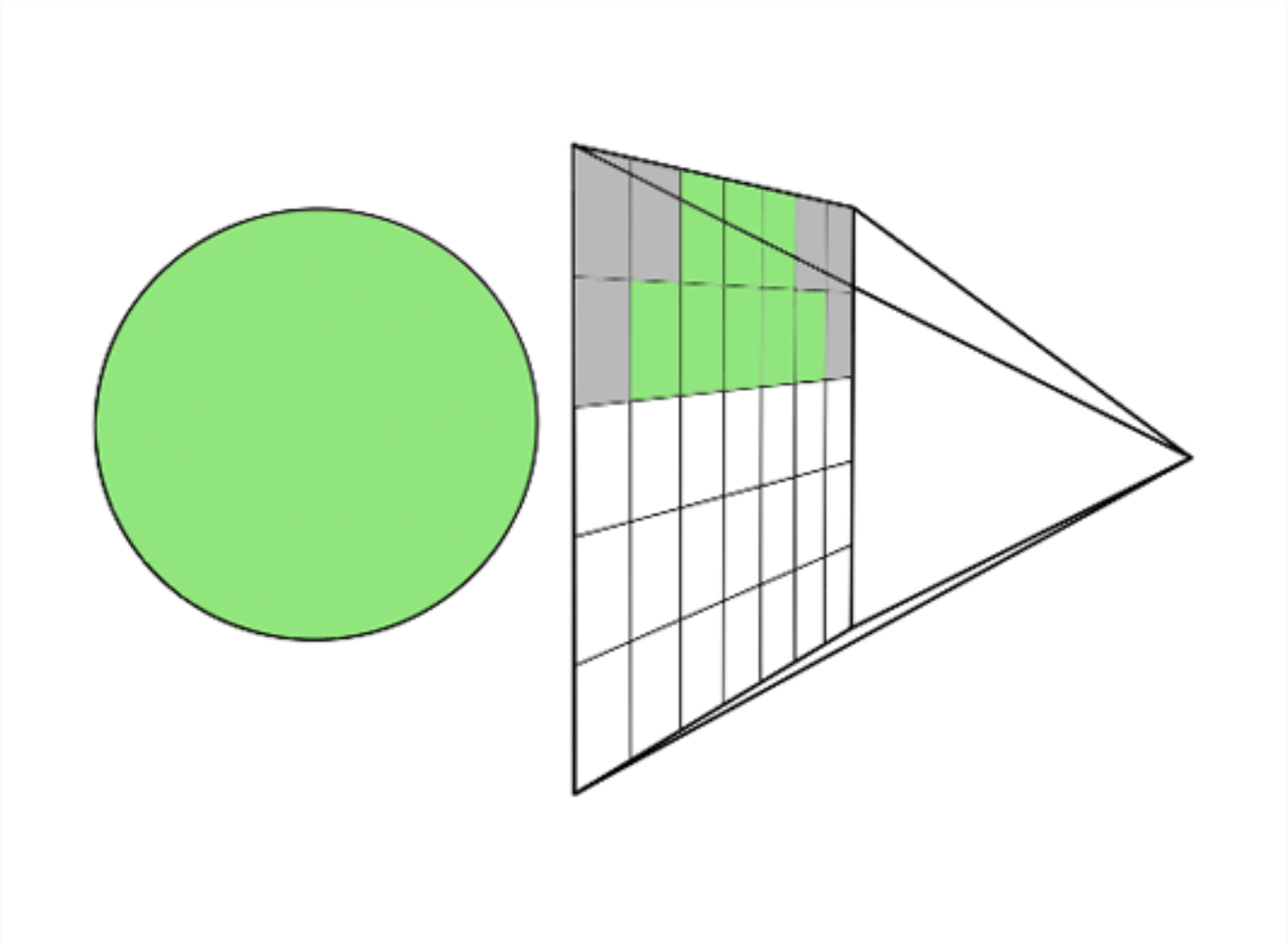
Raycasting Anim



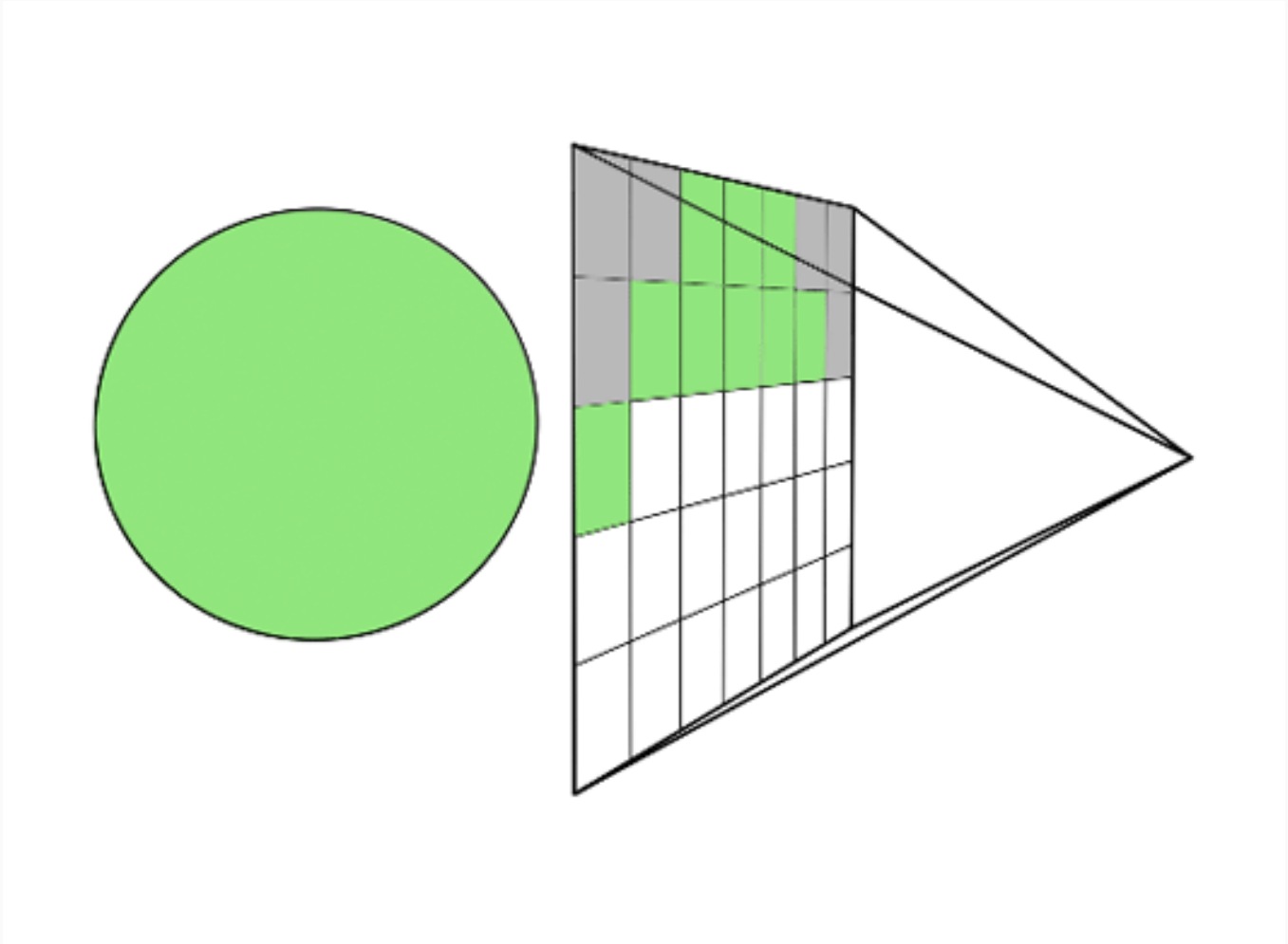
Raycasting Anim



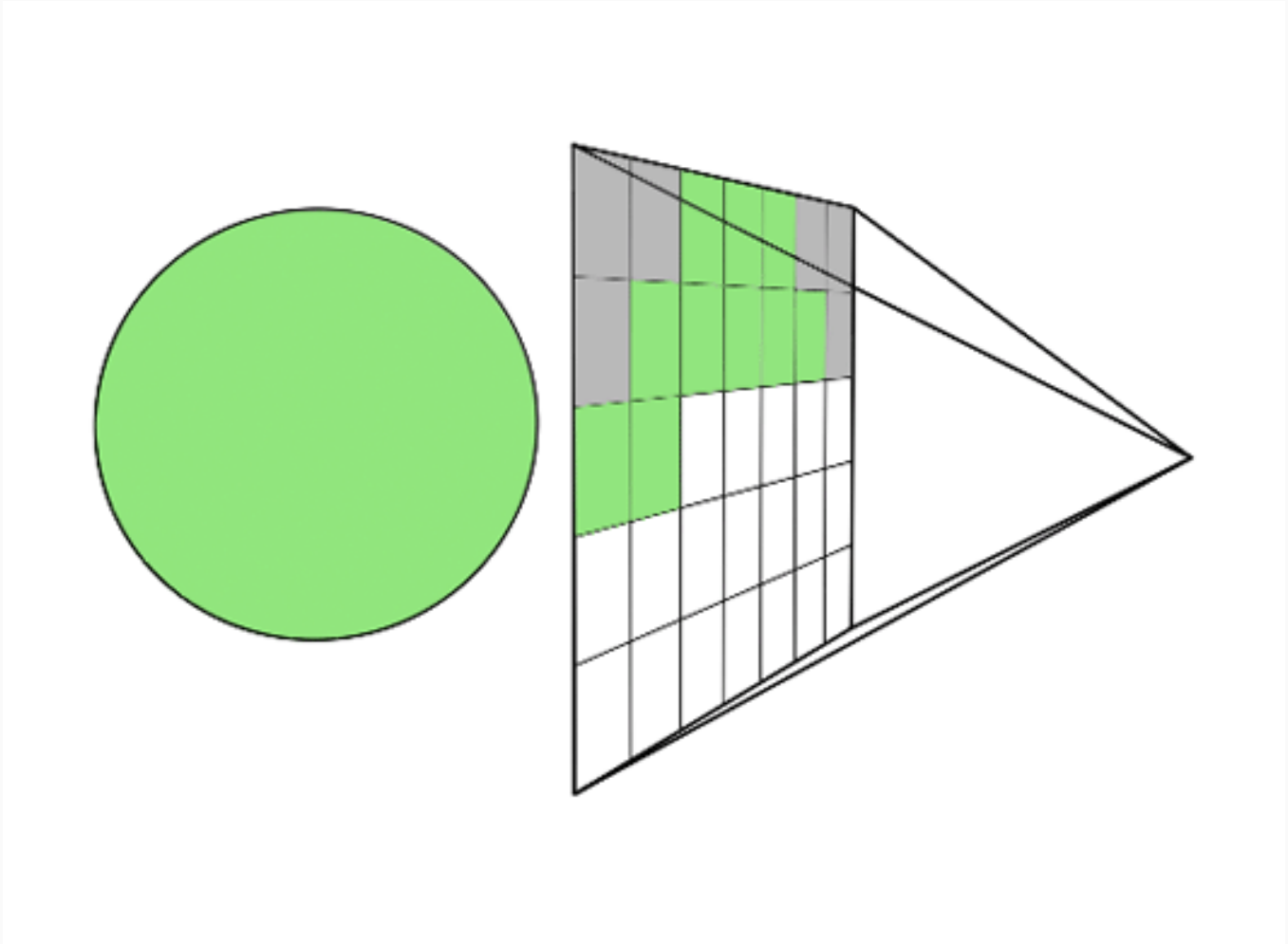
Raycasting Anim



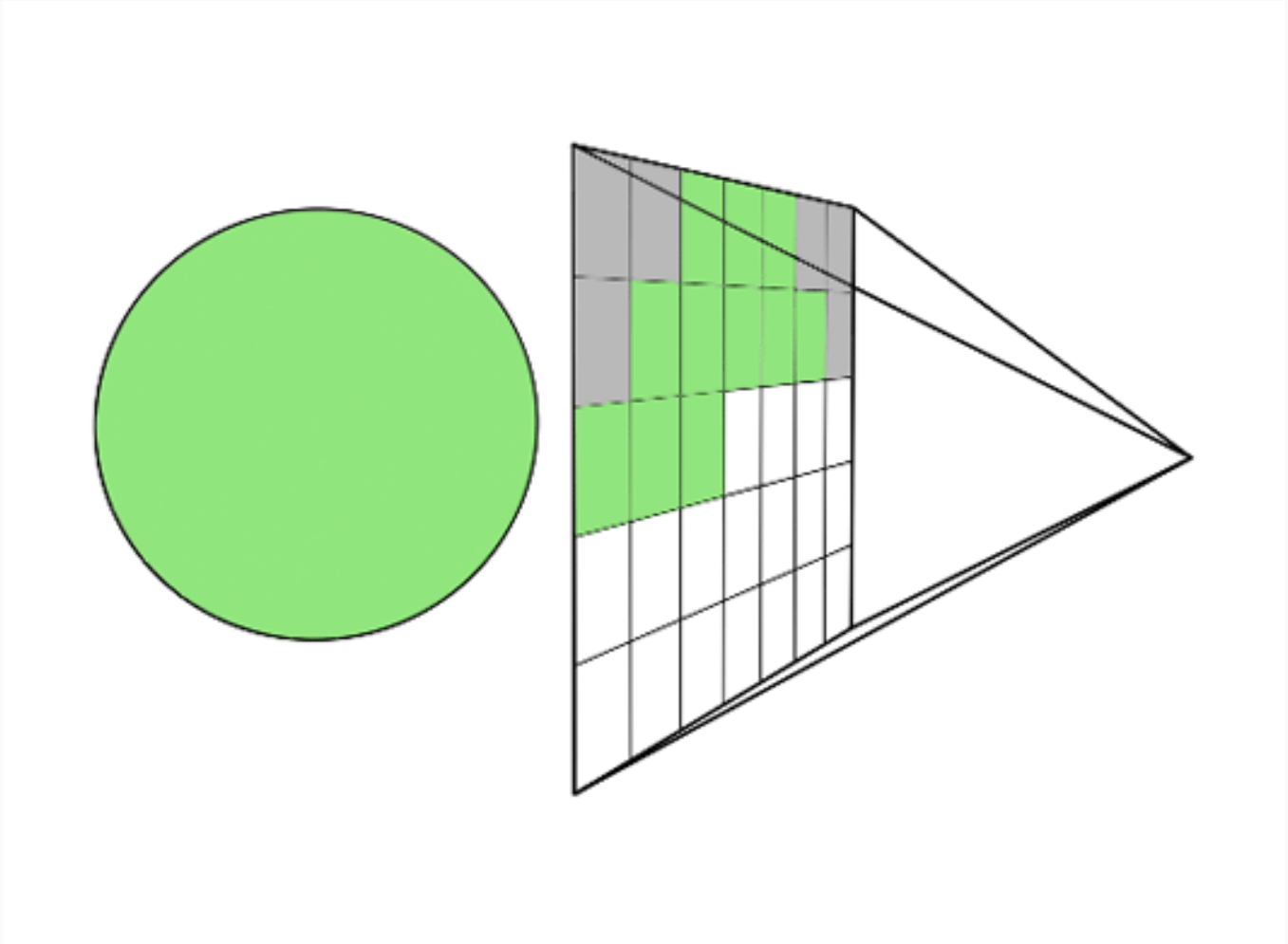
Raycasting Anim



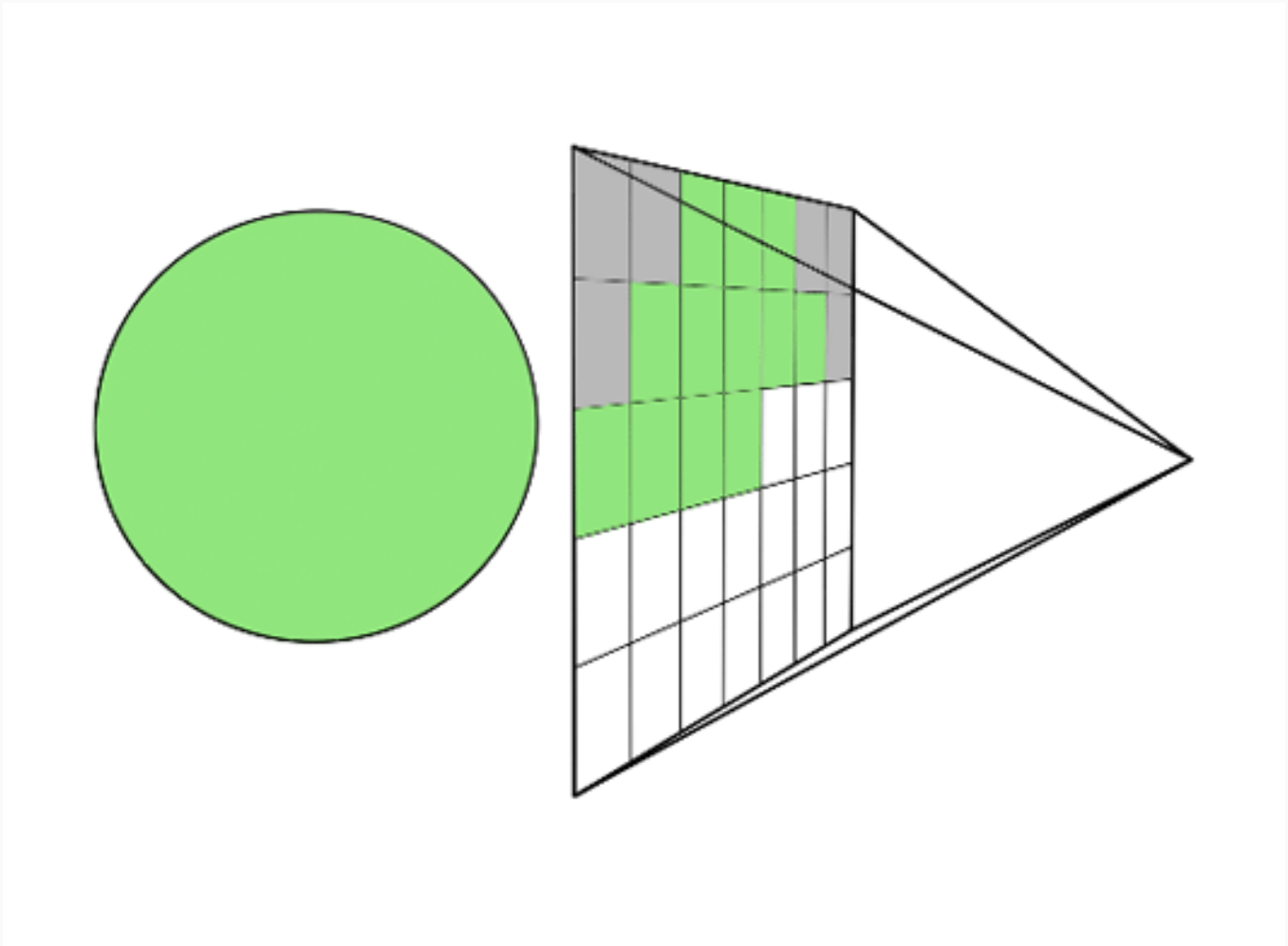
Raycasting Anim



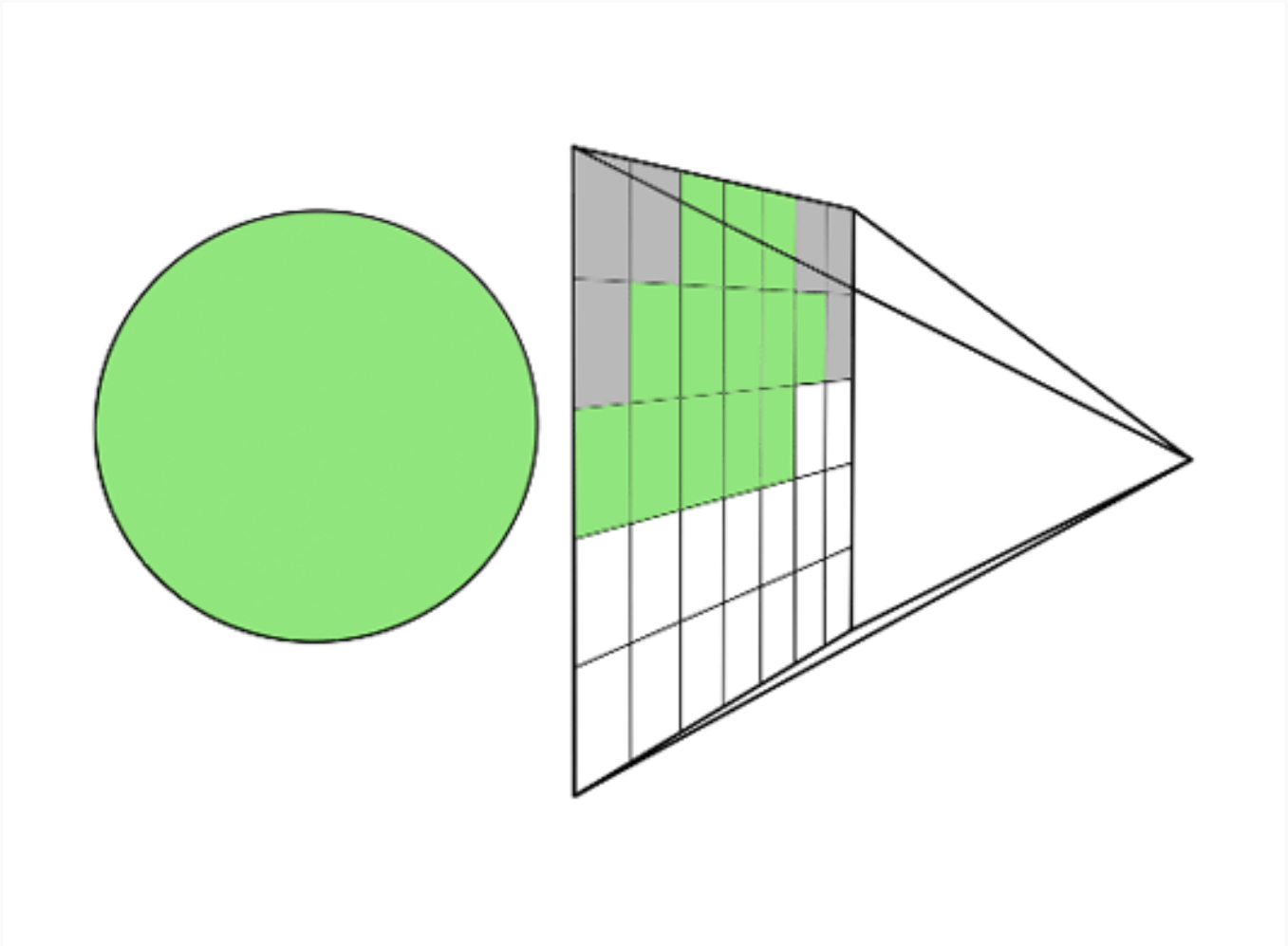
Raycasting Anim



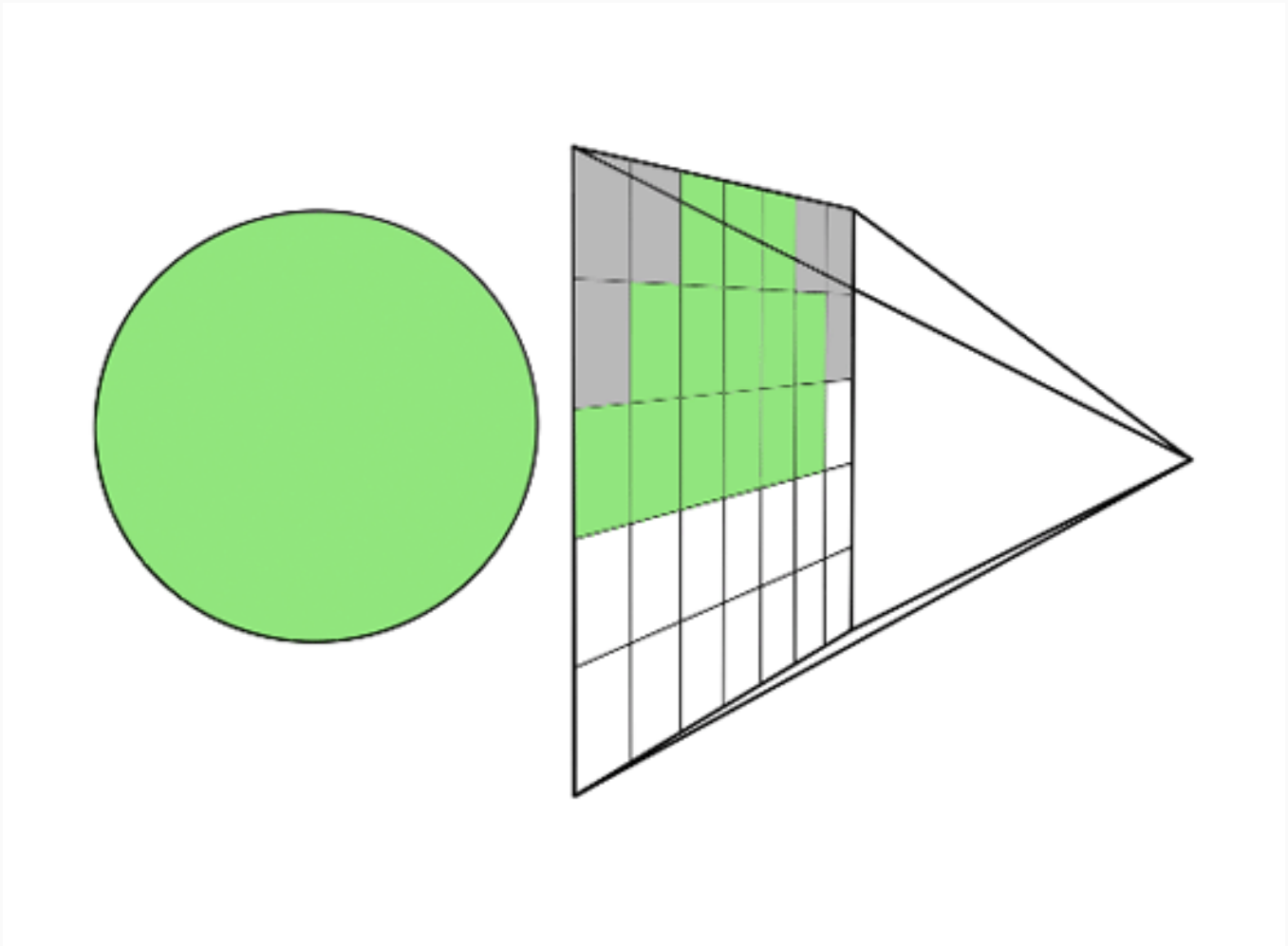
Raycasting Anim



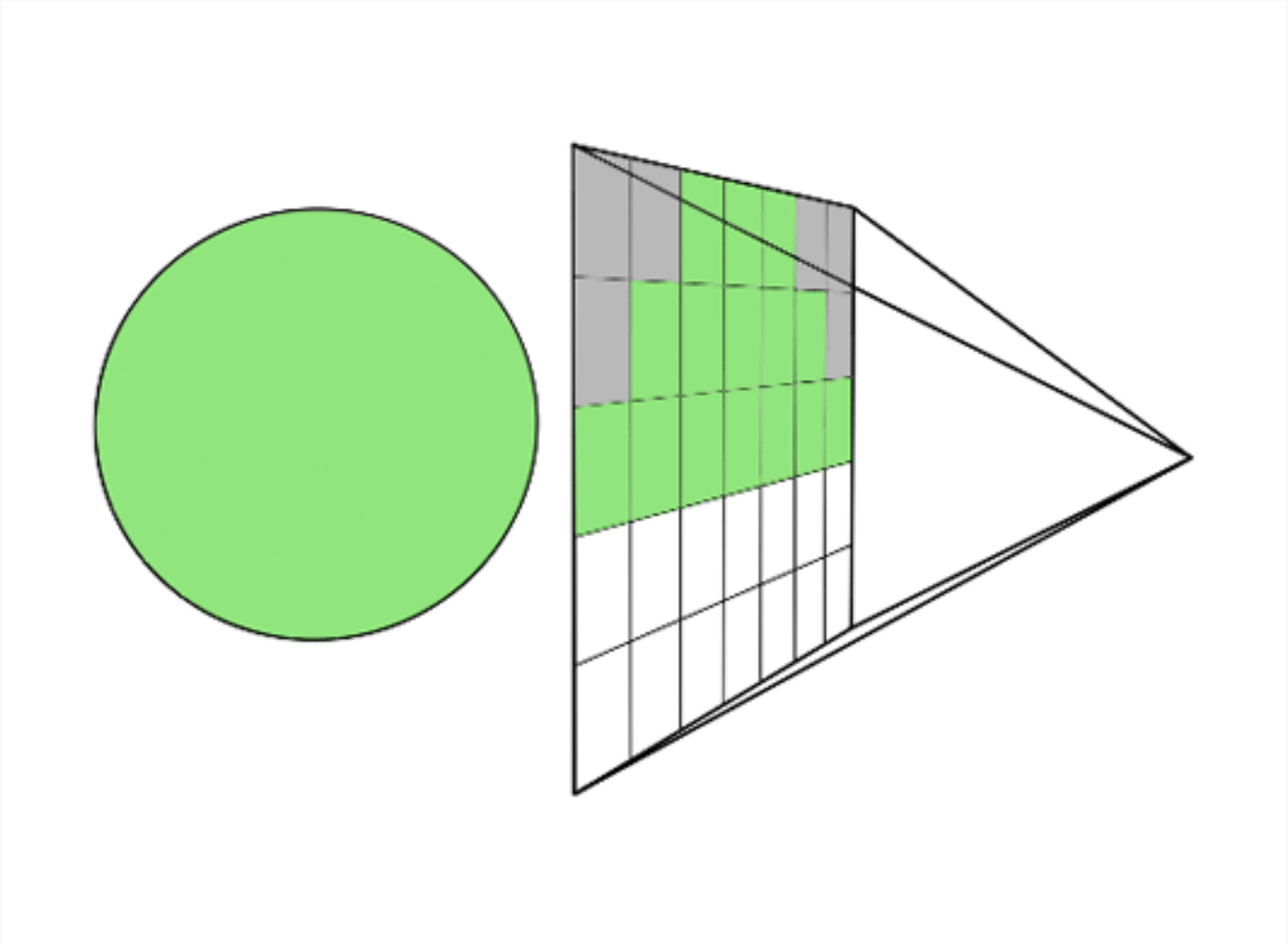
Raycasting Anim



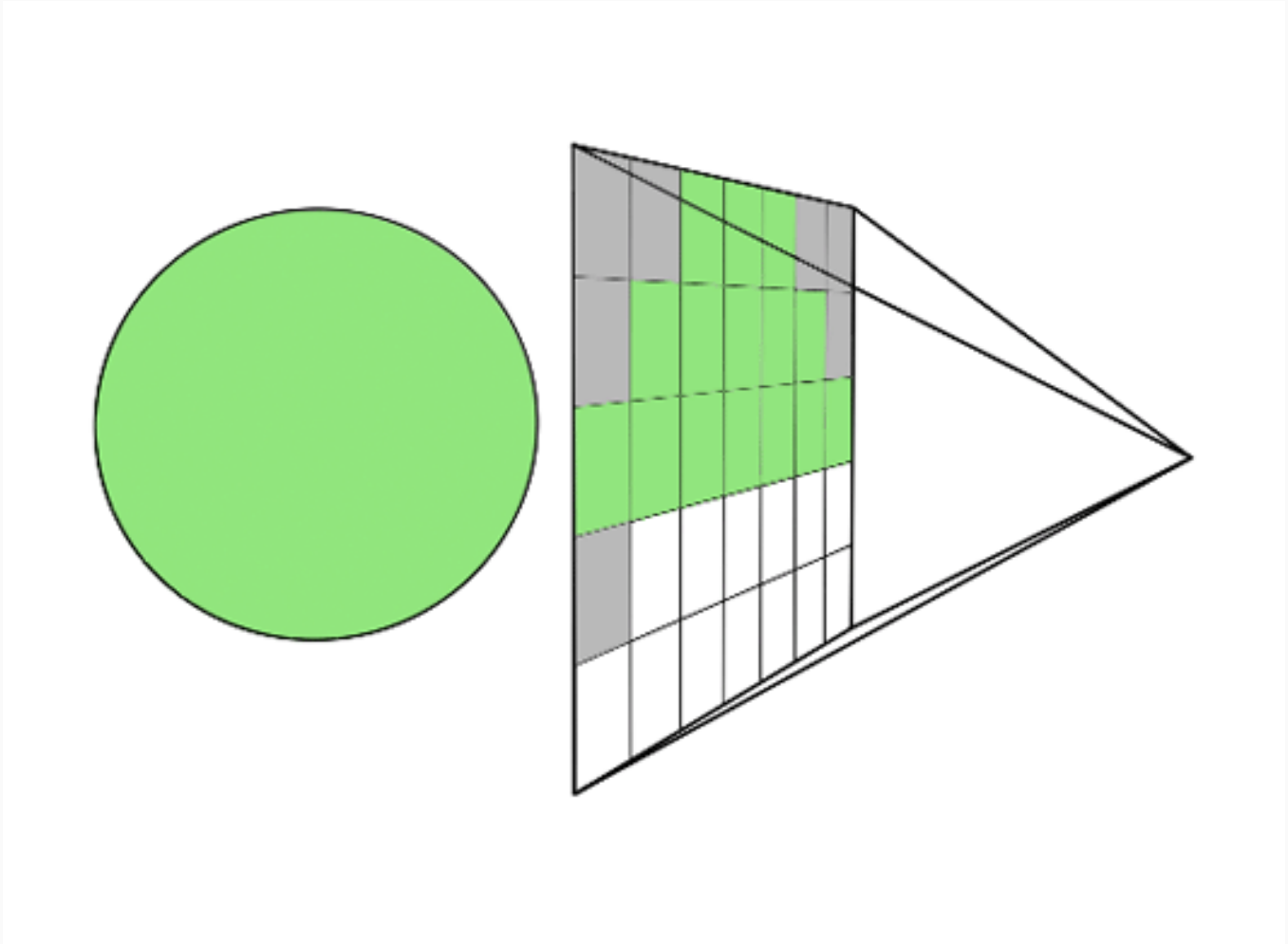
Raycasting Anim



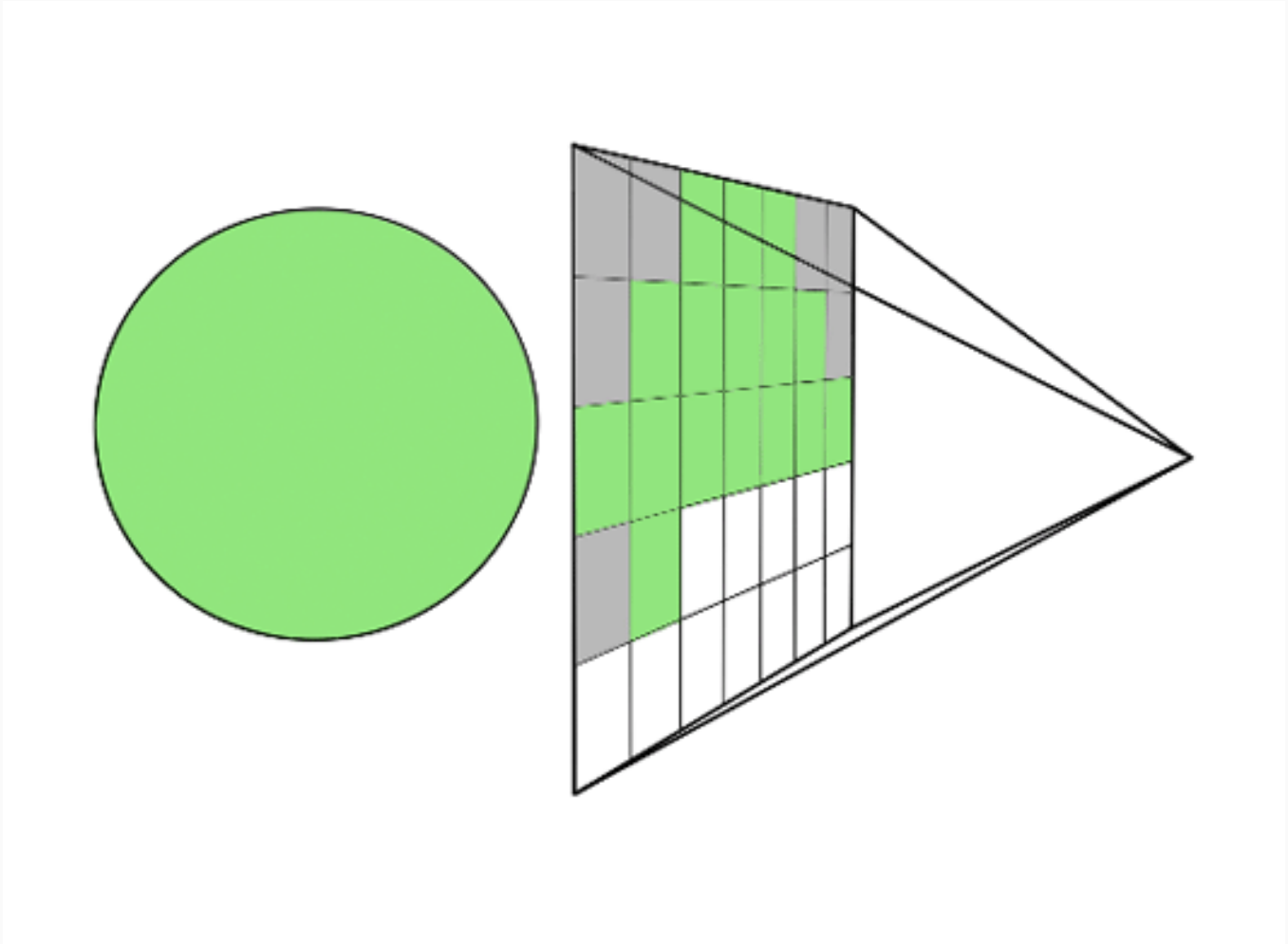
Raycasting Anim



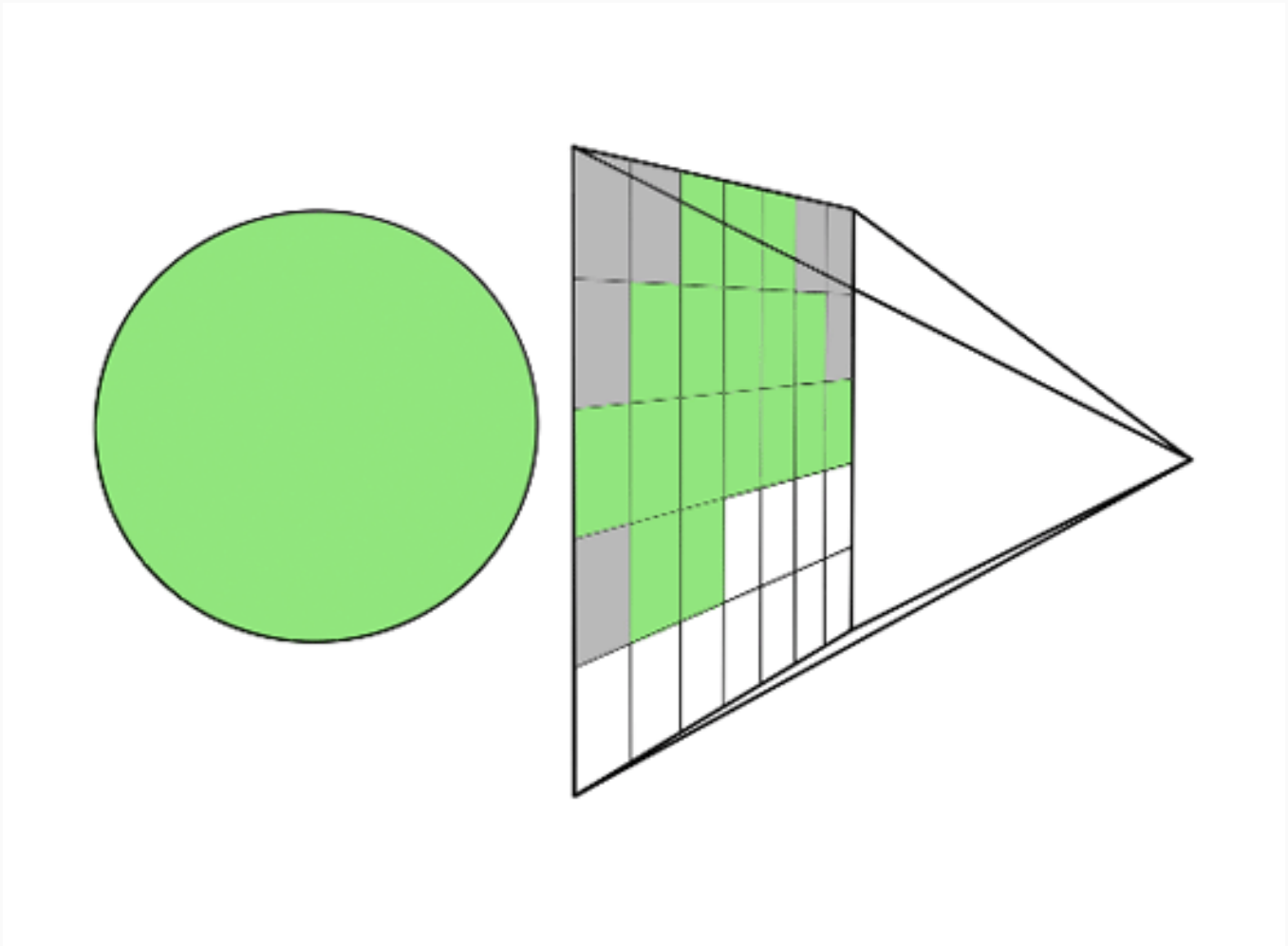
Raycasting Anim



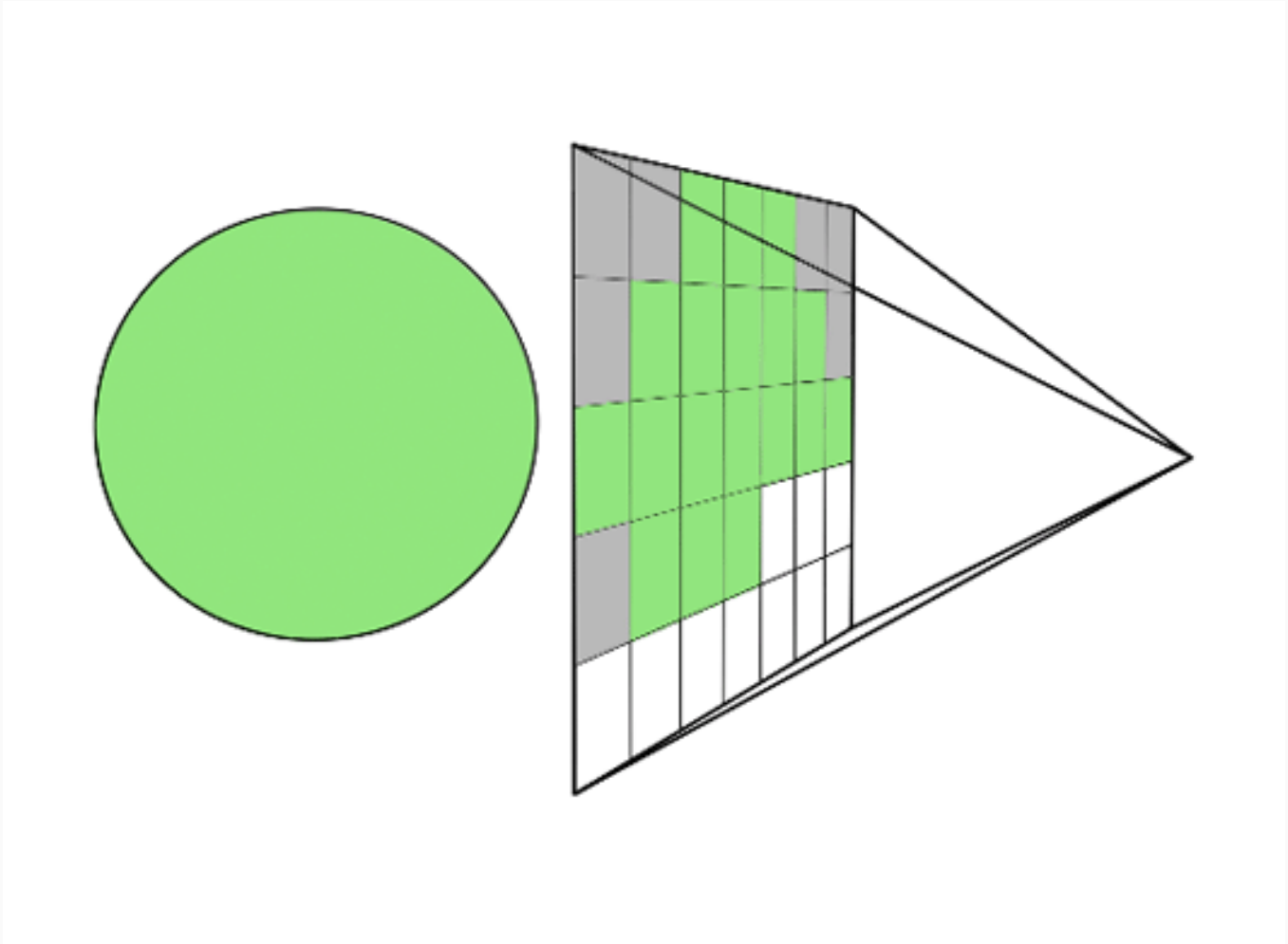
Raycasting Anim



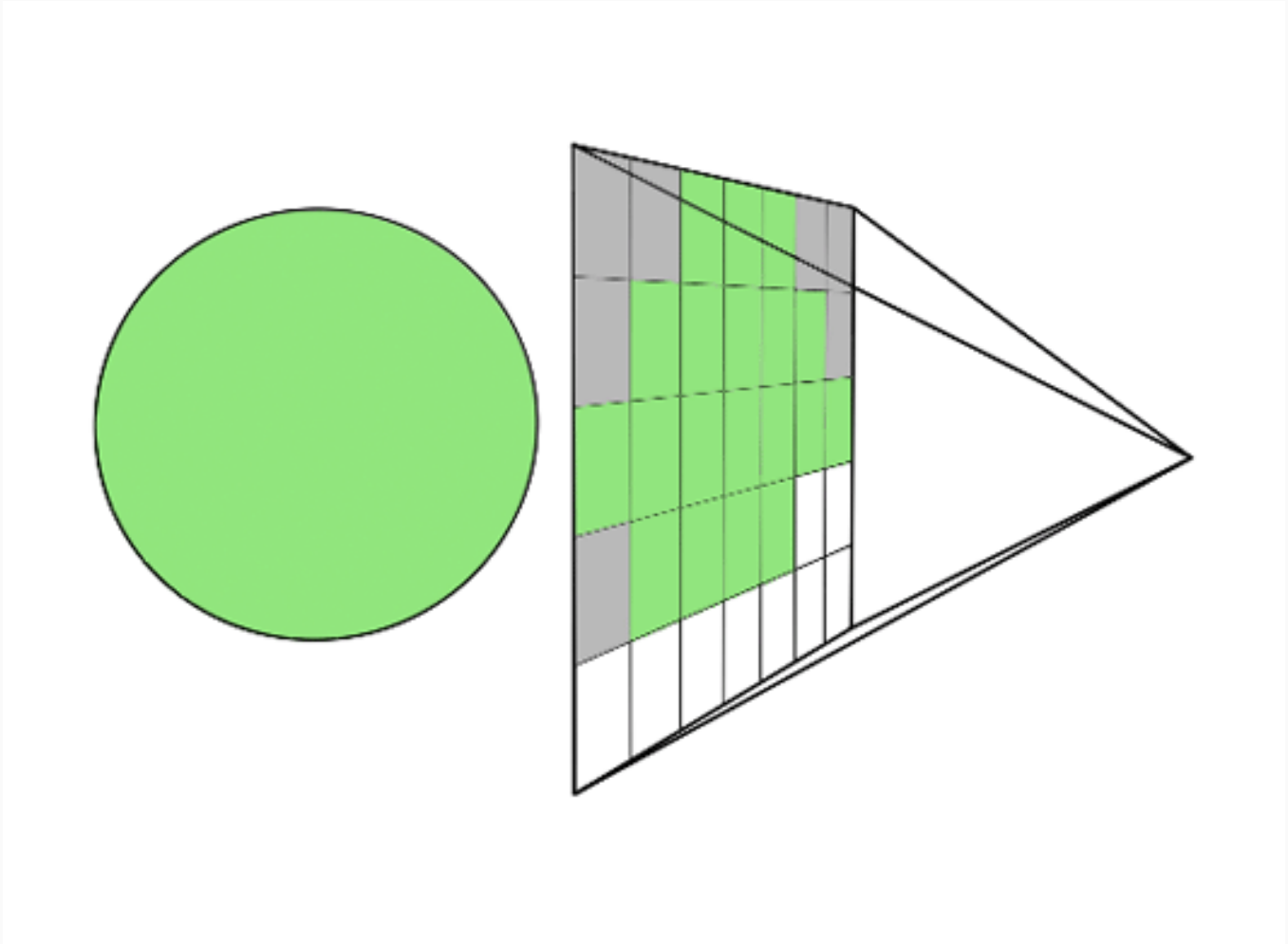
Raycasting Anim



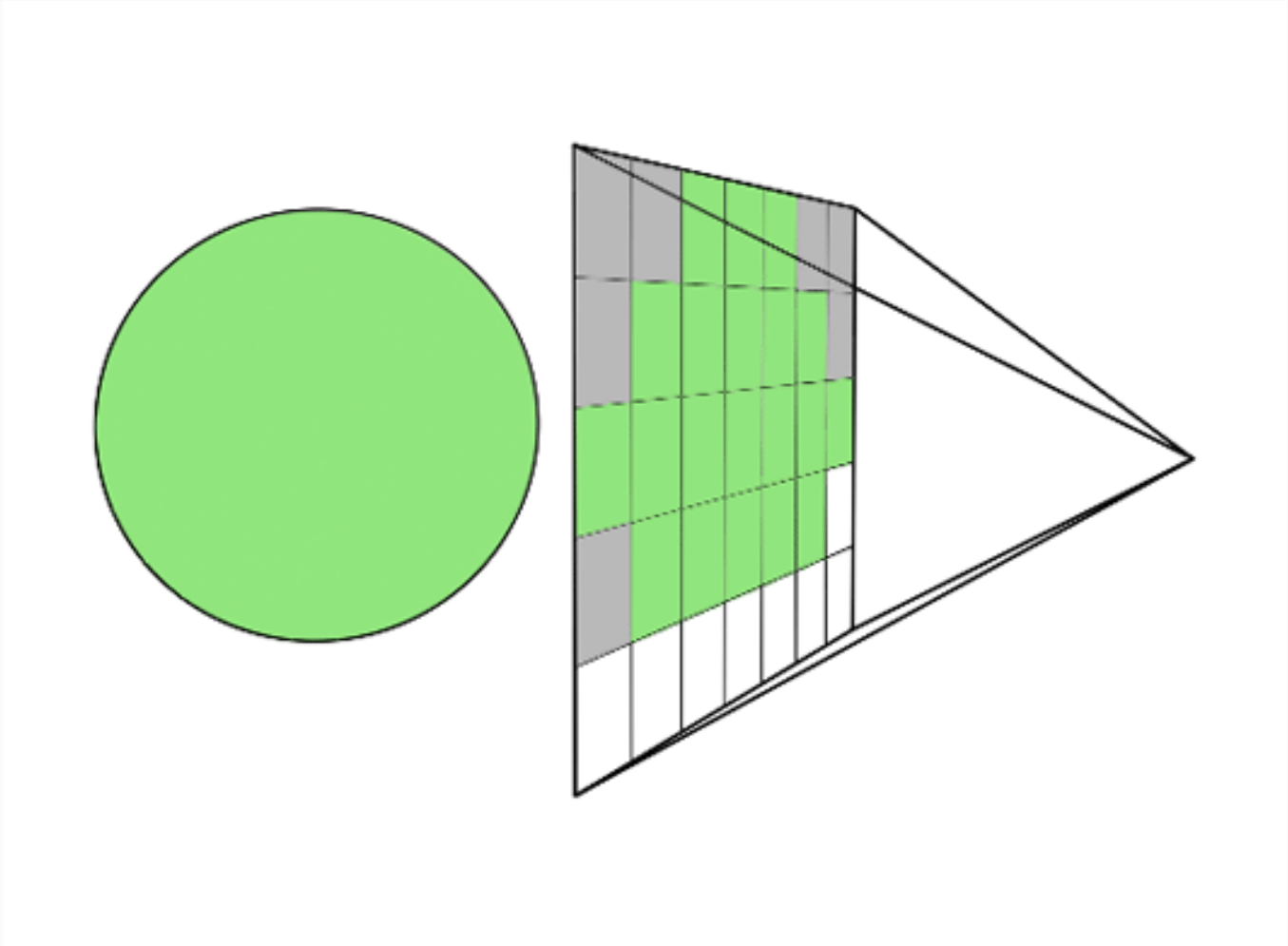
Raycasting Anim



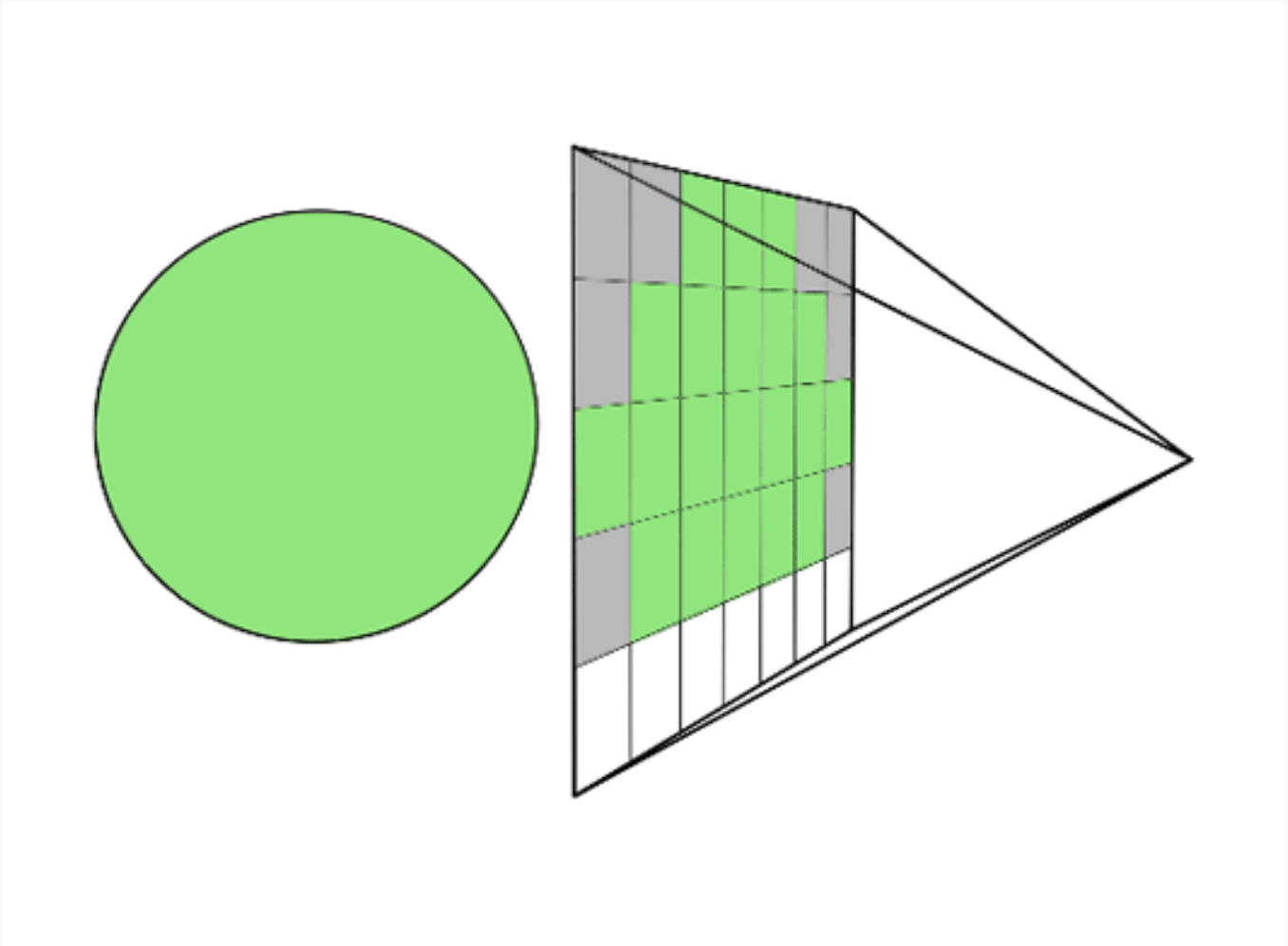
Raycasting Anim



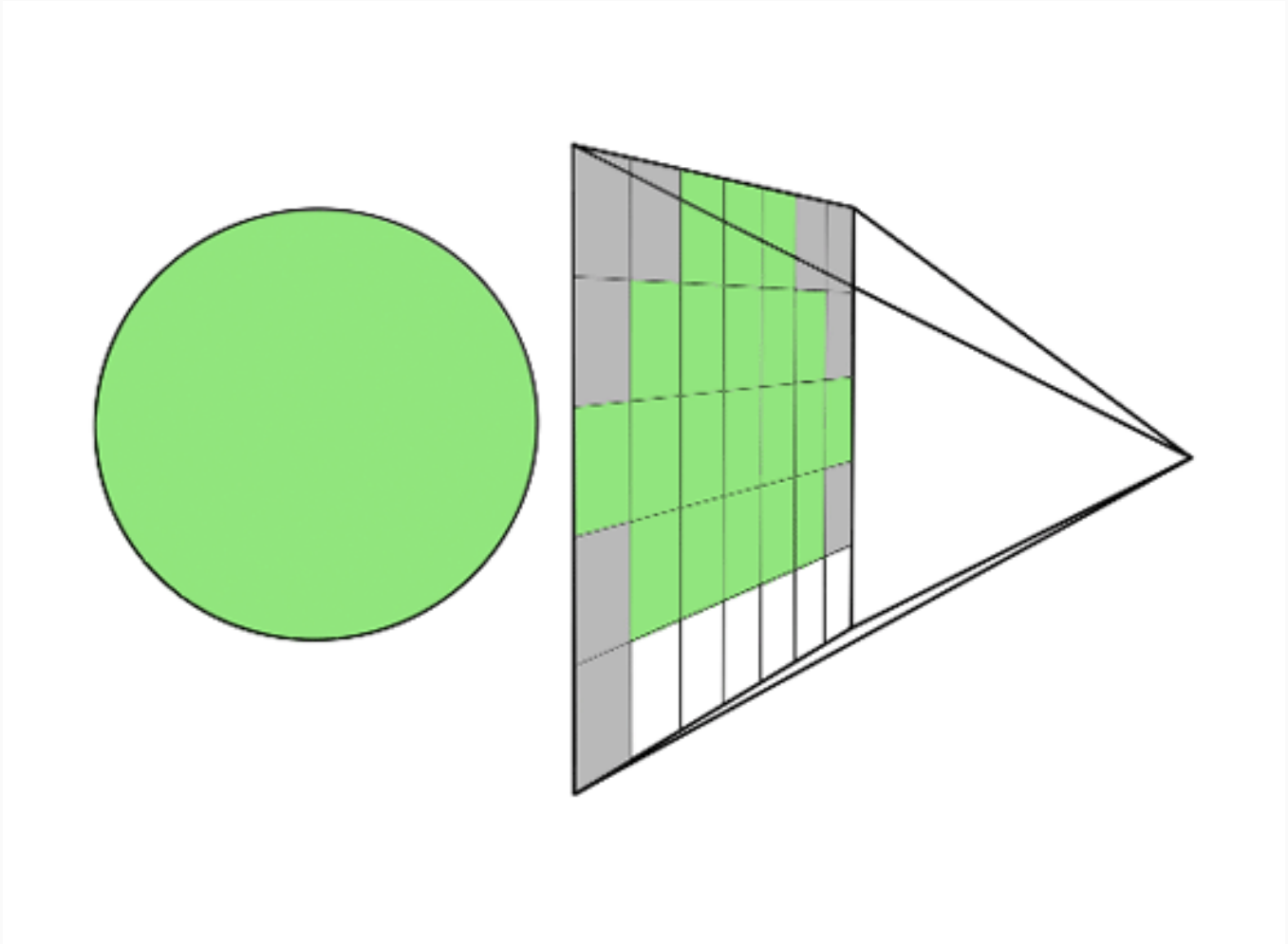
Raycasting Anim



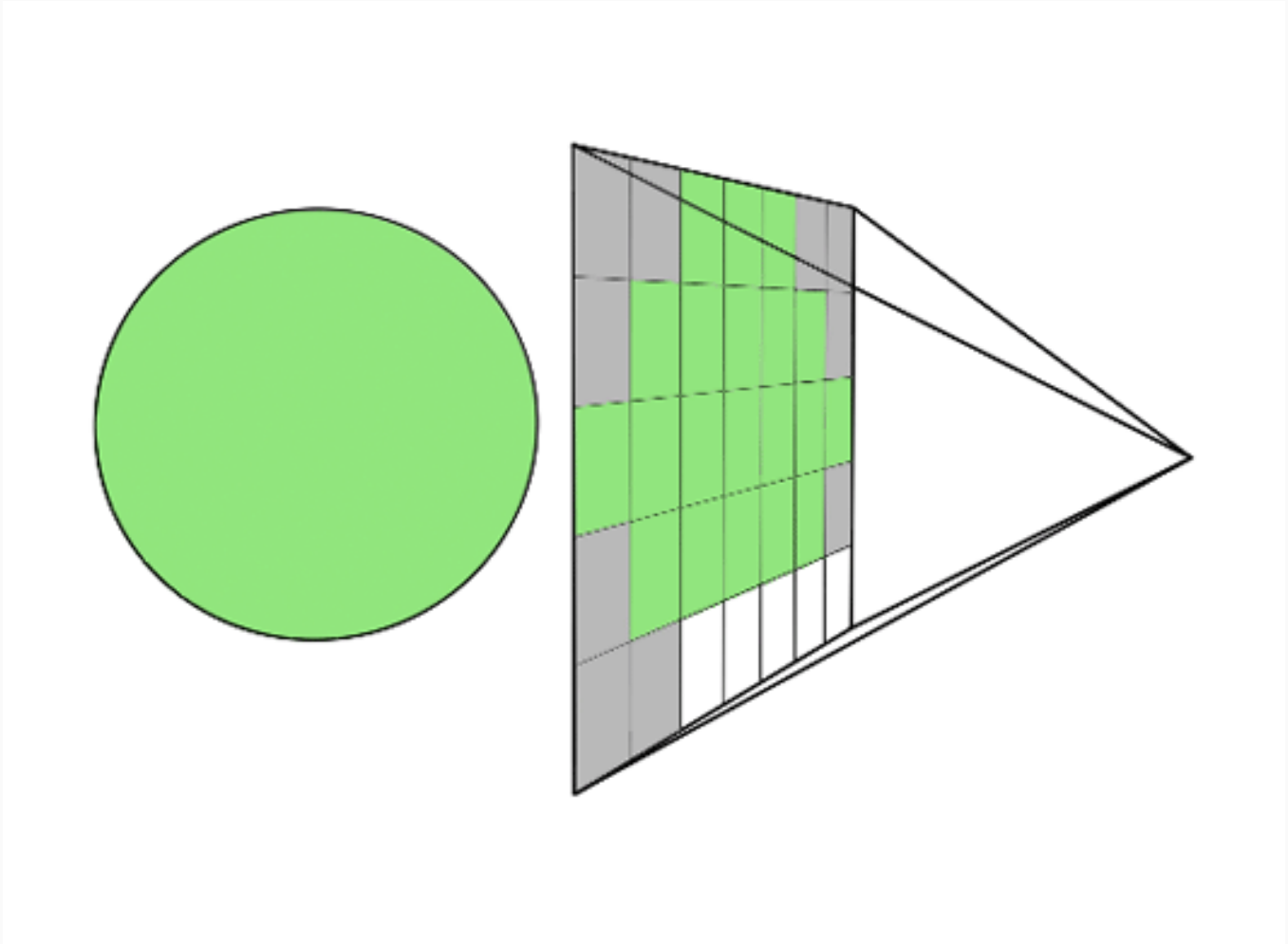
Raycasting Anim



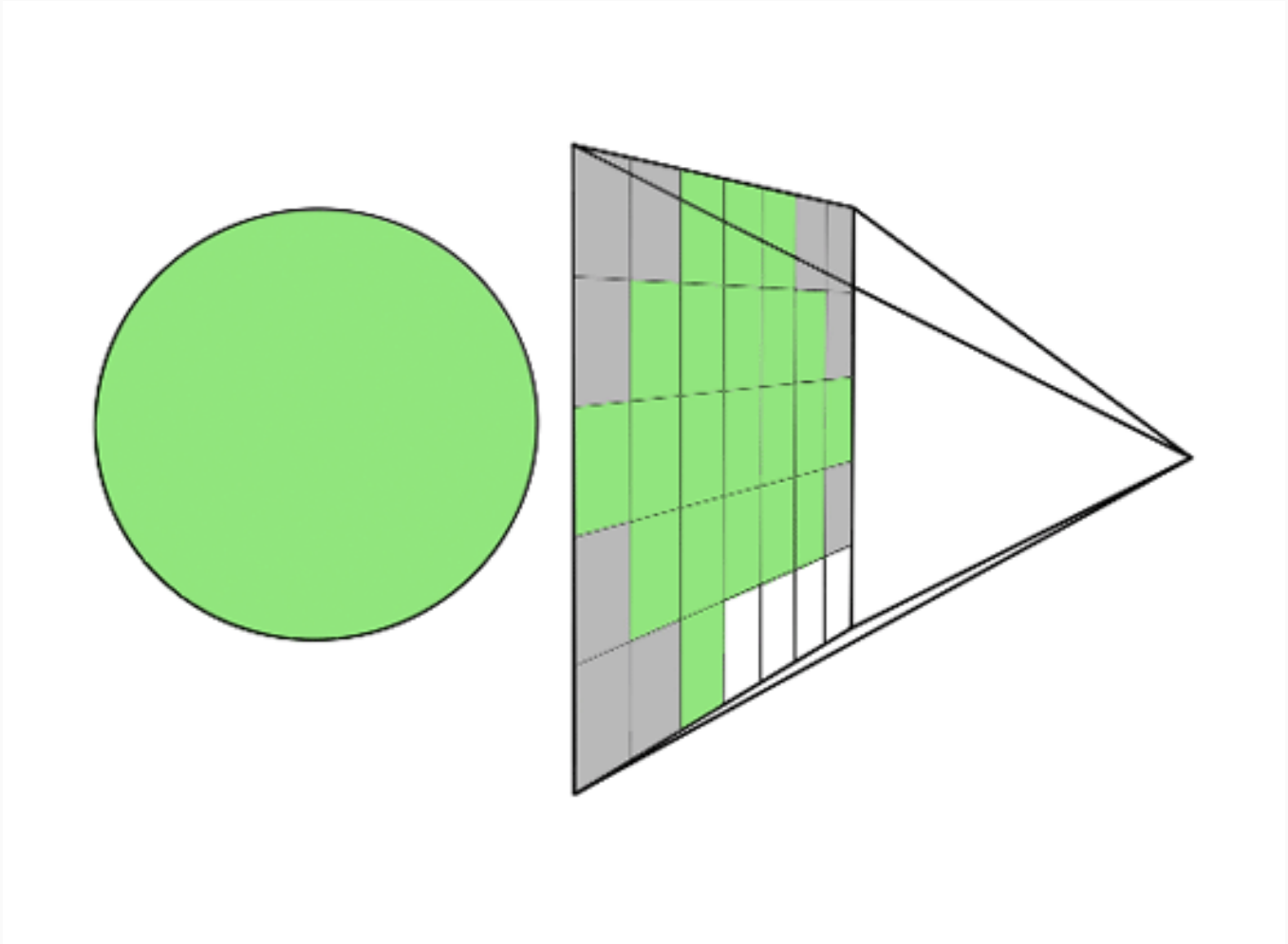
Raycasting Anim



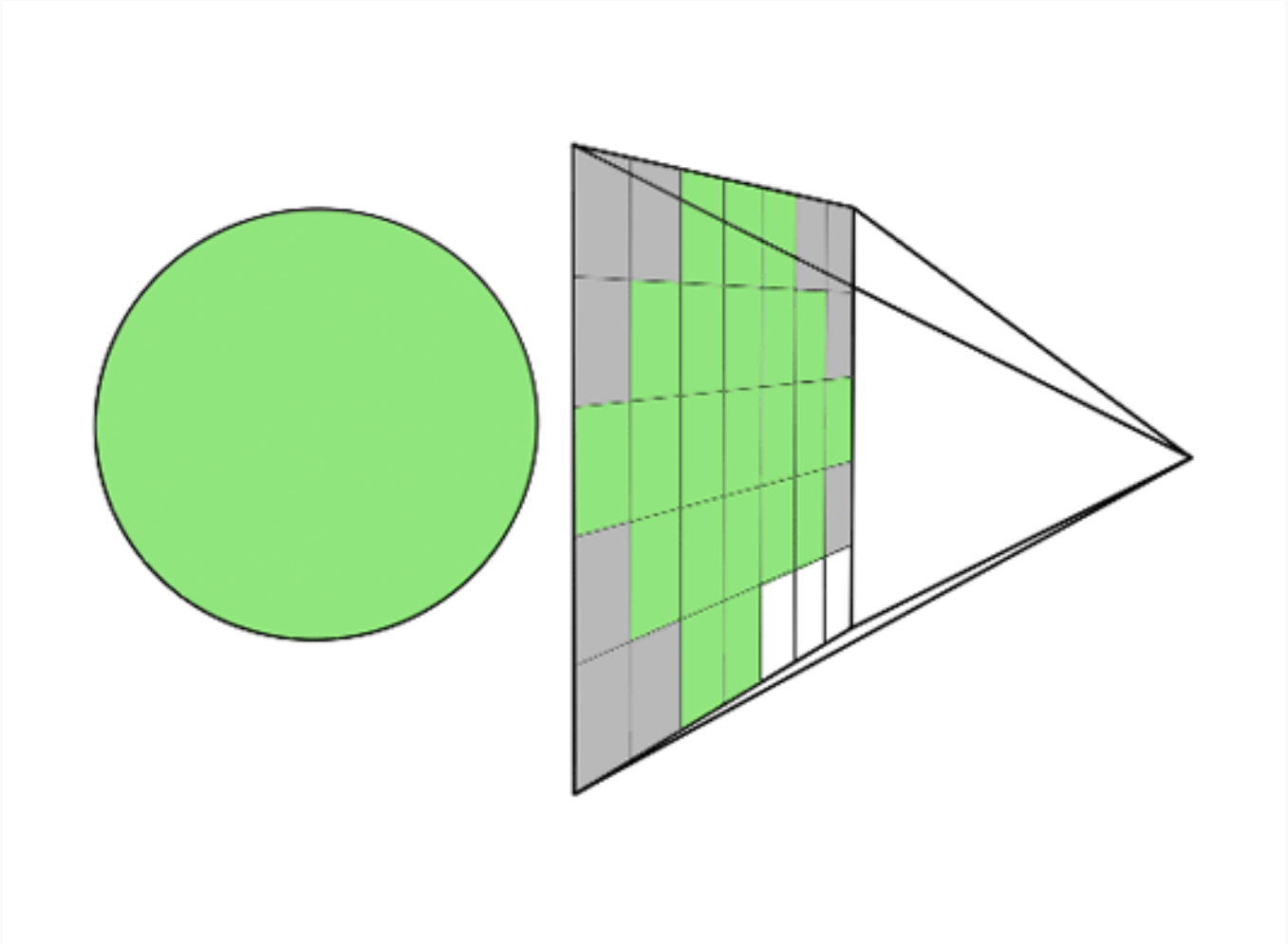
Raycasting Anim



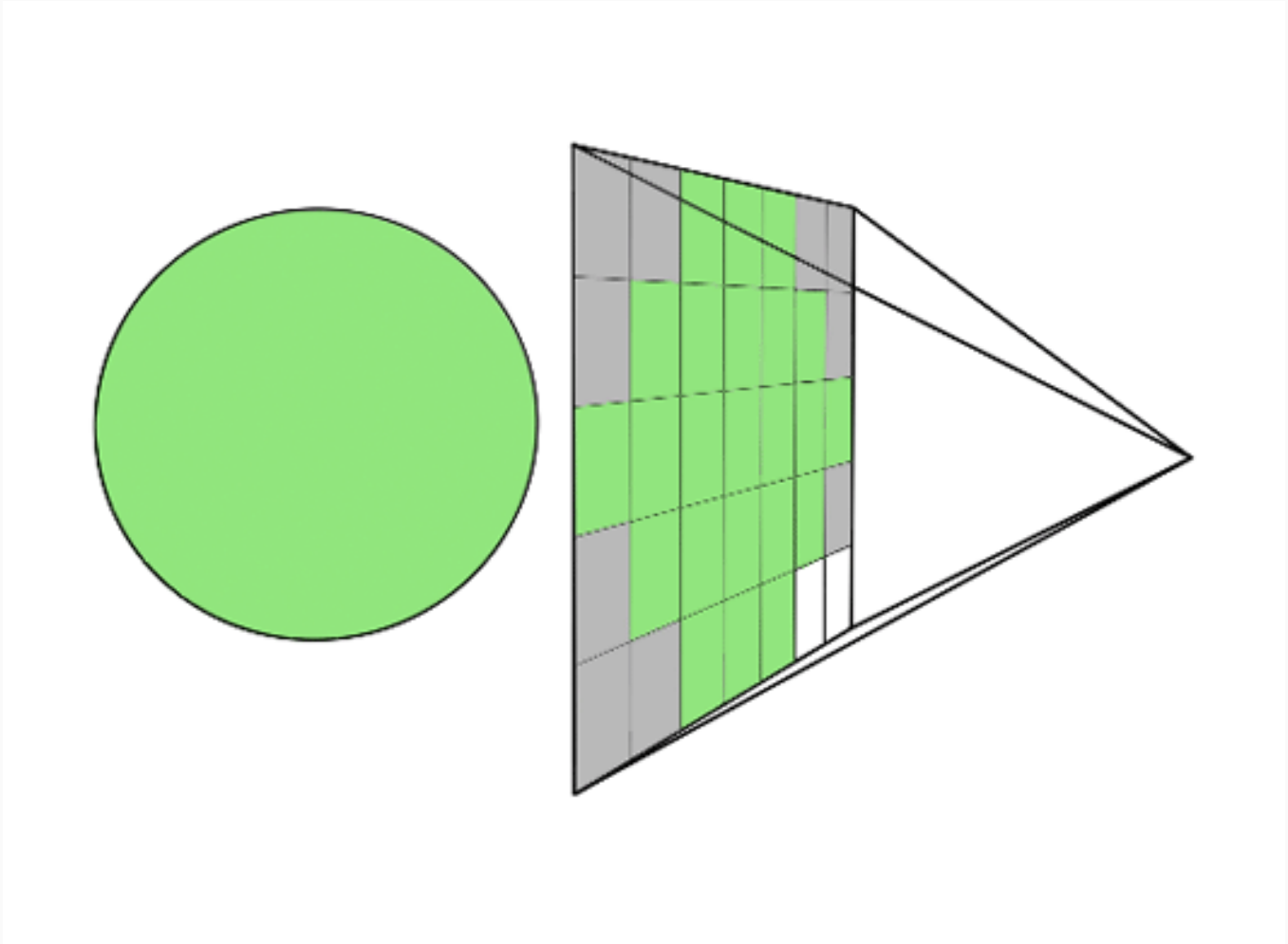
Raycasting Anim



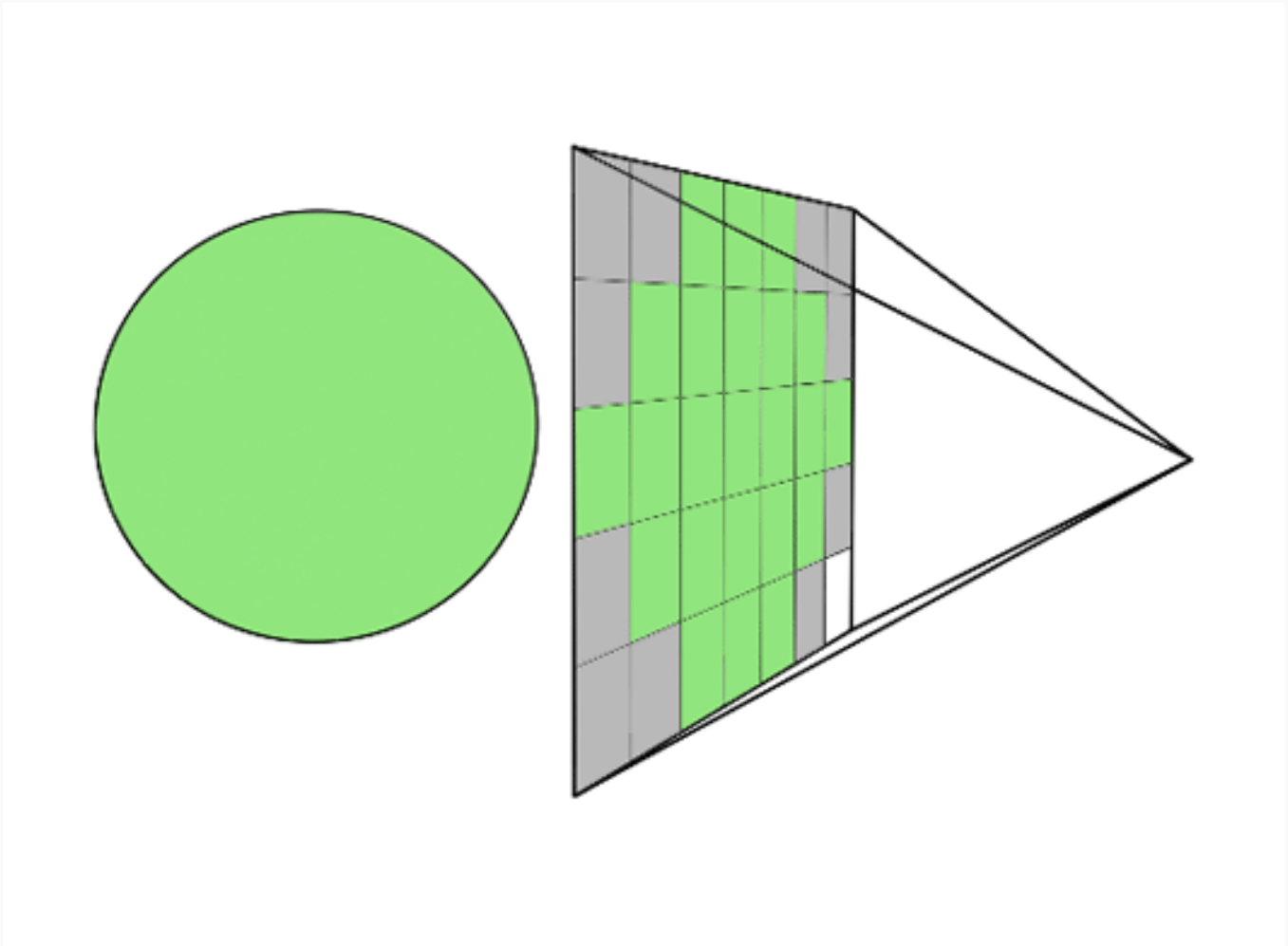
Raycasting Anim



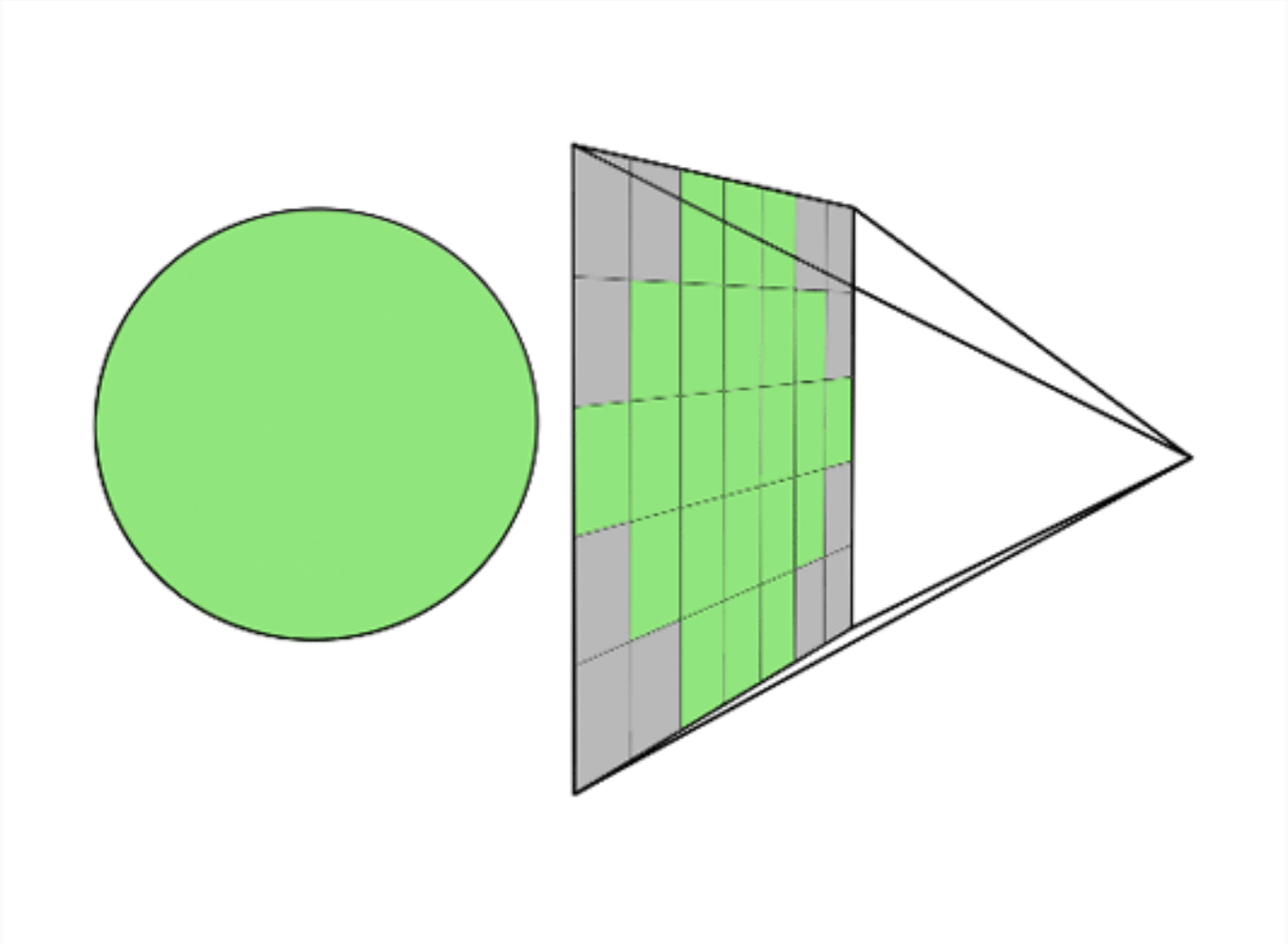
Raycasting Anim



Raycasting Anim



Raycasting Anim



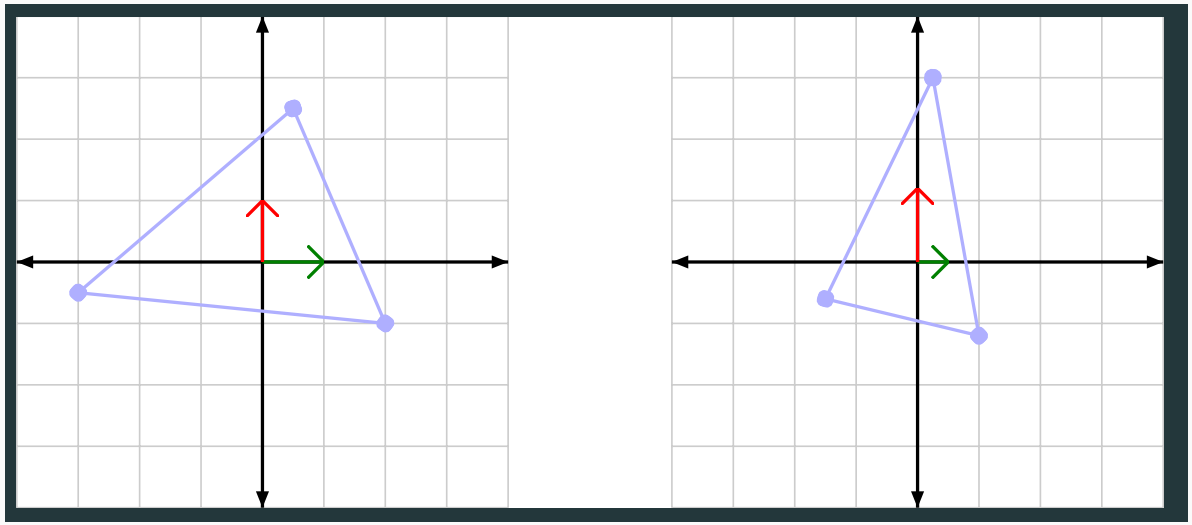
Linear Algebra

Vector Spaces

Point in vector space does not make sense without basis

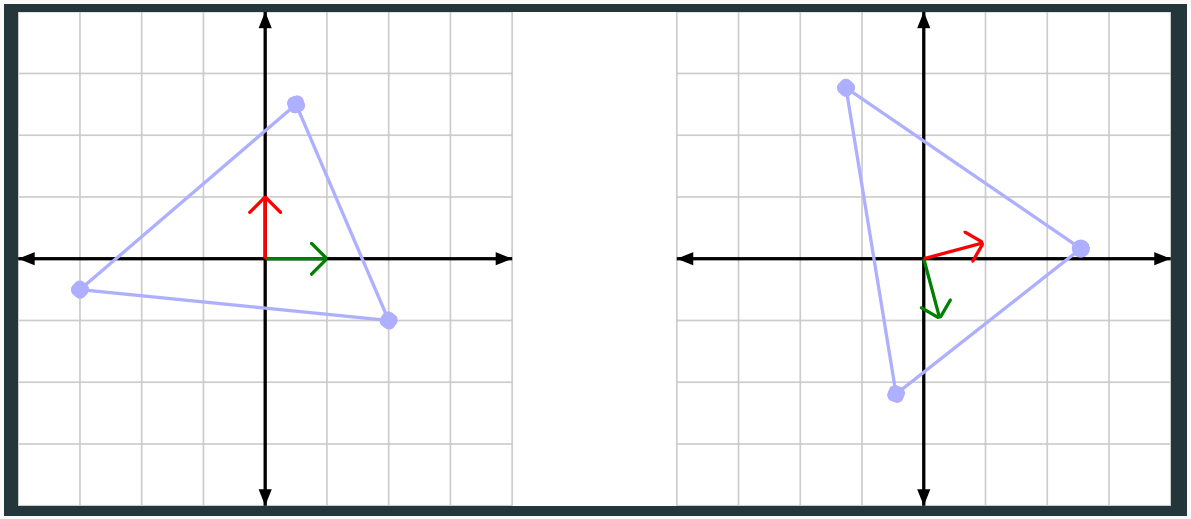
$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3] \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Scaling



$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = [v_x \mathbf{e}_1, v_y \mathbf{e}_2, v_z \mathbf{e}_3] \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & v_z \end{bmatrix} [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3] \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Rotation



$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \mathbf{R}\left[\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\right] \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Rotation

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

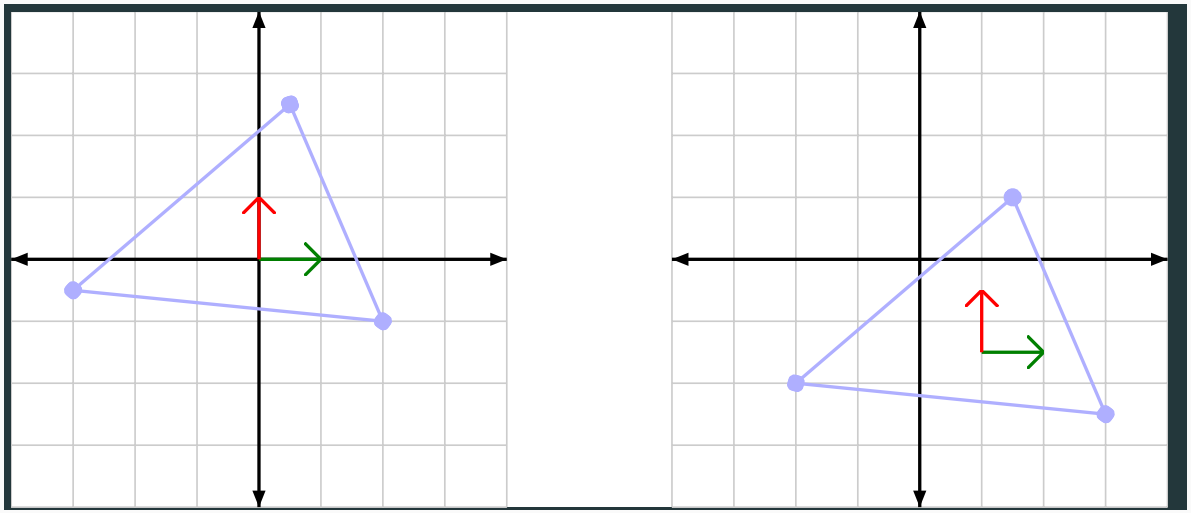
- Matrix multiplication is not commutative
  - $\mathbf{AB} \neq \mathbf{BA}$
  - Need to determine order convention
- Pick and order and stick with it

Rotation

$$R_{xyz} = \begin{bmatrix} \cos\theta_y \cos\theta_z & -\cos\theta_x \sin\theta_z + \sin\theta_x \sin\theta_y \cos\theta_z & \sin\theta_x \sin\theta_z + \cos\theta_x \sin\theta_y \cos\theta_z \\ \cos\theta_y \sin\theta_z & \cos\theta_x \cos\theta_z + \sin\theta_x \sin\theta_y \sin\theta_z & -\sin\theta_x \cos\theta_z + \cos\theta_x \sin\theta_y \sin\theta_z \\ -\sin\theta_y & \sin\theta_x \cos\theta_y & \cos\theta_x \cos\theta_y \end{bmatrix}$$

- Matrix multiplication is not commutative
  - $\mathbf{AB} \neq \mathbf{BA}$
  - Need to determine order convention
- Pick and order and stick with it

Translation



$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3] \begin{bmatrix} x + dx \\ y + dy \\ z + dz \end{bmatrix}$$

Homogenous Coordinates[5]

- We do not want to deal with translation differently compared to other transformations
- We can work in *projective* or *homogenous* coordinate representations instead
- We will append the space with one dimension and specify an equivivalence class of all projections

Homogenous Coordinates

- Perspective Projection

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \end{bmatrix}$$

- Specify equivivalence class between all projected points

$$\begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \sim \begin{bmatrix} 4 \\ 6 \\ 8 \end{bmatrix}$$

- Homogenous coordinates

$$\begin{bmatrix} \frac{2}{4} \\ \frac{3}{4} \\ \frac{4}{4} \end{bmatrix} = \begin{bmatrix} \frac{4}{8} \\ \frac{6}{8} \\ \frac{8}{8} \end{bmatrix} = \begin{bmatrix} \frac{2}{4} \\ \frac{3}{4} \\ 1 \end{bmatrix}$$

Homogenous Coordinates

- Cartesian 3D space

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Allows for representing points in infinity

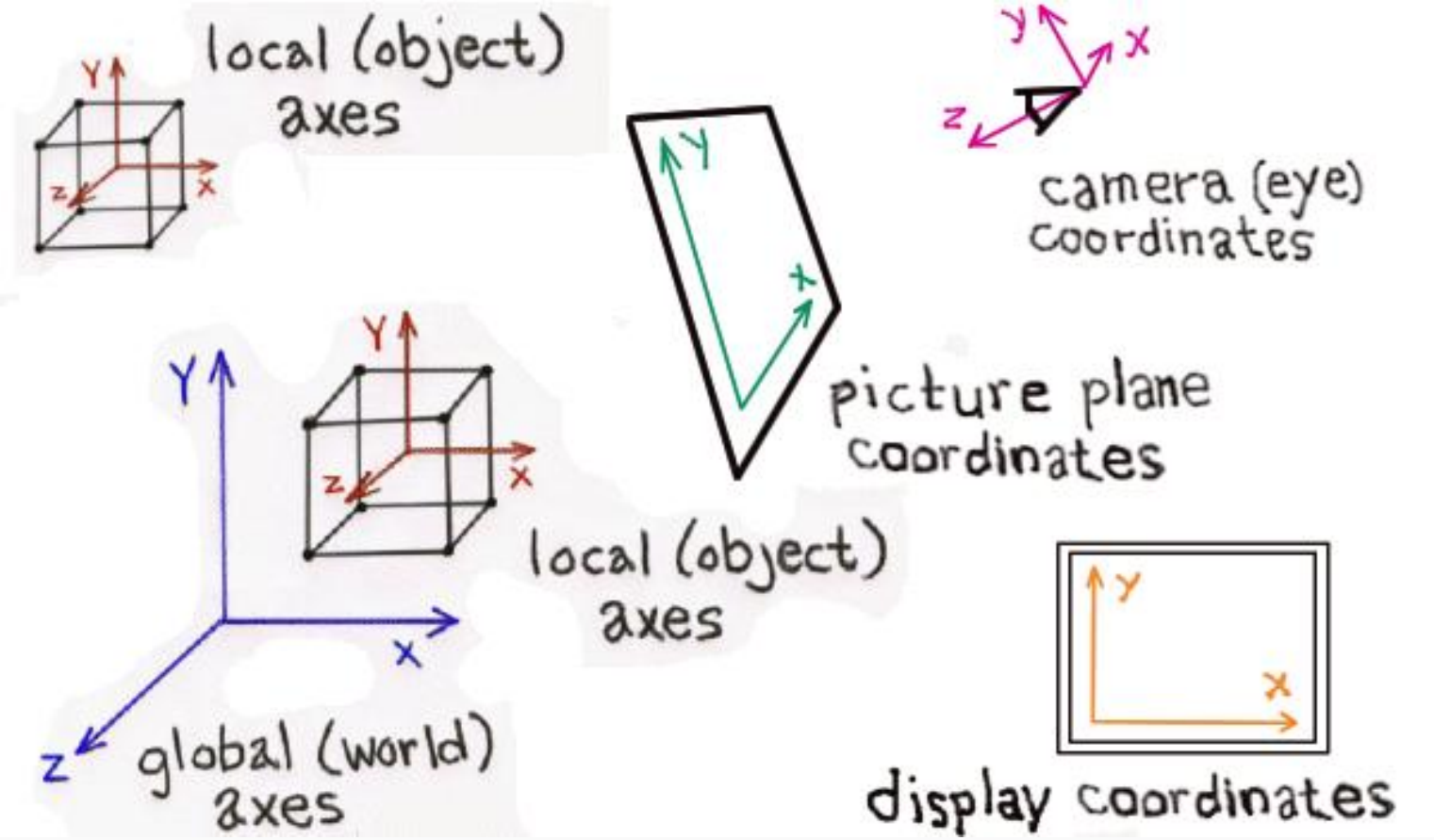
$$\lim_{d \to \infty} \begin{bmatrix} x + d \\ y + d \\ z + d \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

- For example, under projection two parallell lines will meet

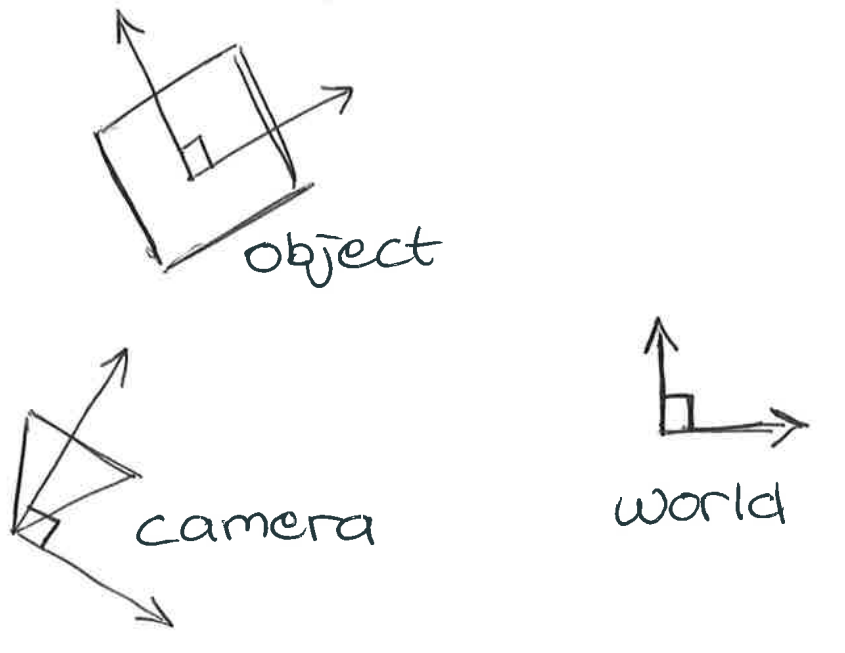
Homogenous Coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} v_x \cdot R_{11} & R_{12} & R_{13} & dx \\ R_{21} & v_y \cdot R_{22} & R_{23} & dy \\ R_{31} & R_{32} & v_z \cdot R_{33} & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Cordinate Transforms



View Transformations



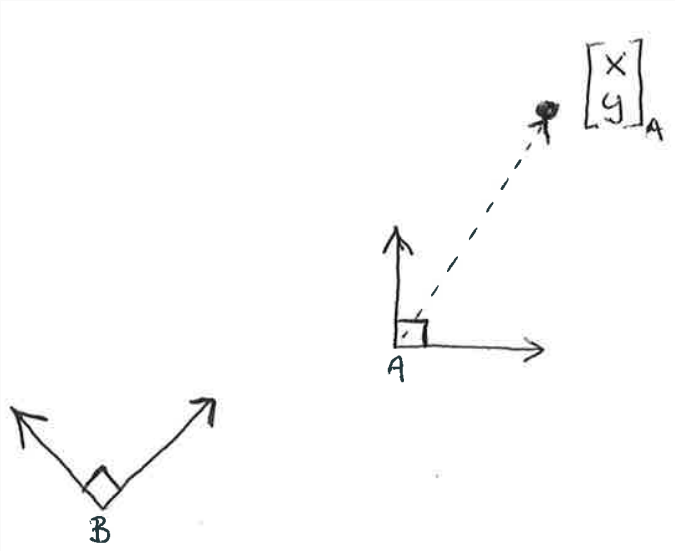
View Transformations

1. Translate

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}_A - \begin{bmatrix} x \\ y \end{bmatrix}_{O_B}$$

2. Rotate (negative)

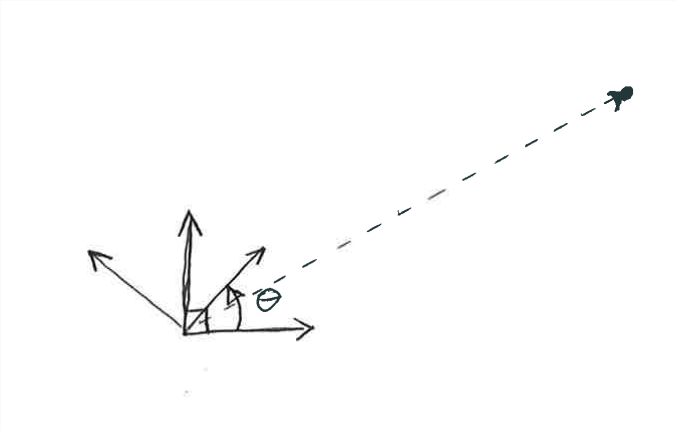
$$\begin{bmatrix} x \\ y \end{bmatrix}_B = \begin{bmatrix} cos(-\theta) & -sin(-\theta) \\ sin(-\theta) & cos(-\theta) \end{bmatrix}_B \begin{bmatrix} x' \\ y' \end{bmatrix}$$



View Transformations

1. Translate

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}_A - \begin{bmatrix} x \\ y \end{bmatrix}_{O_B}$$



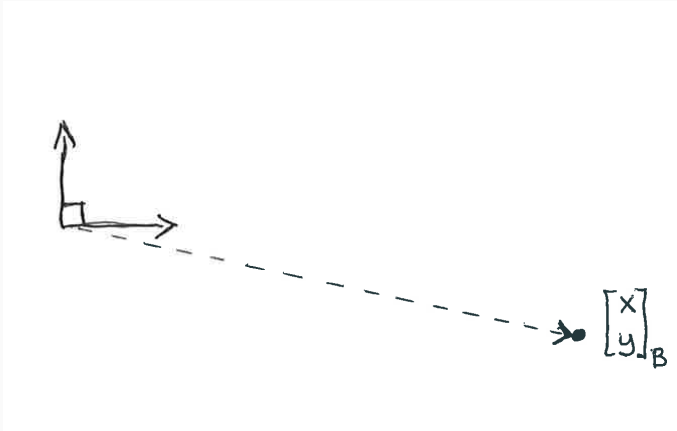
2. Rotate (negative)

$$\begin{bmatrix} x \\ y \end{bmatrix}_B = \begin{bmatrix} cos(-\theta) & -sin(-\theta) \\ sin(-\theta) & cos(-\theta) \end{bmatrix}_B \begin{bmatrix} x' \\ y' \end{bmatrix}$$

View Transformations

1. Translate

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}_A - \begin{bmatrix} x \\ y \end{bmatrix}_{O_B}$$



2. Rotate (negative)

$$\begin{bmatrix} x \\ y \end{bmatrix}_B = \begin{bmatrix} cos(-\theta) & -sin(-\theta) \\ sin(-\theta) & cos(-\theta) \end{bmatrix}_B \begin{bmatrix} x' \\ y' \end{bmatrix}$$

View Transformations

Transformation

Always the same order for every transformation of coordinate systems

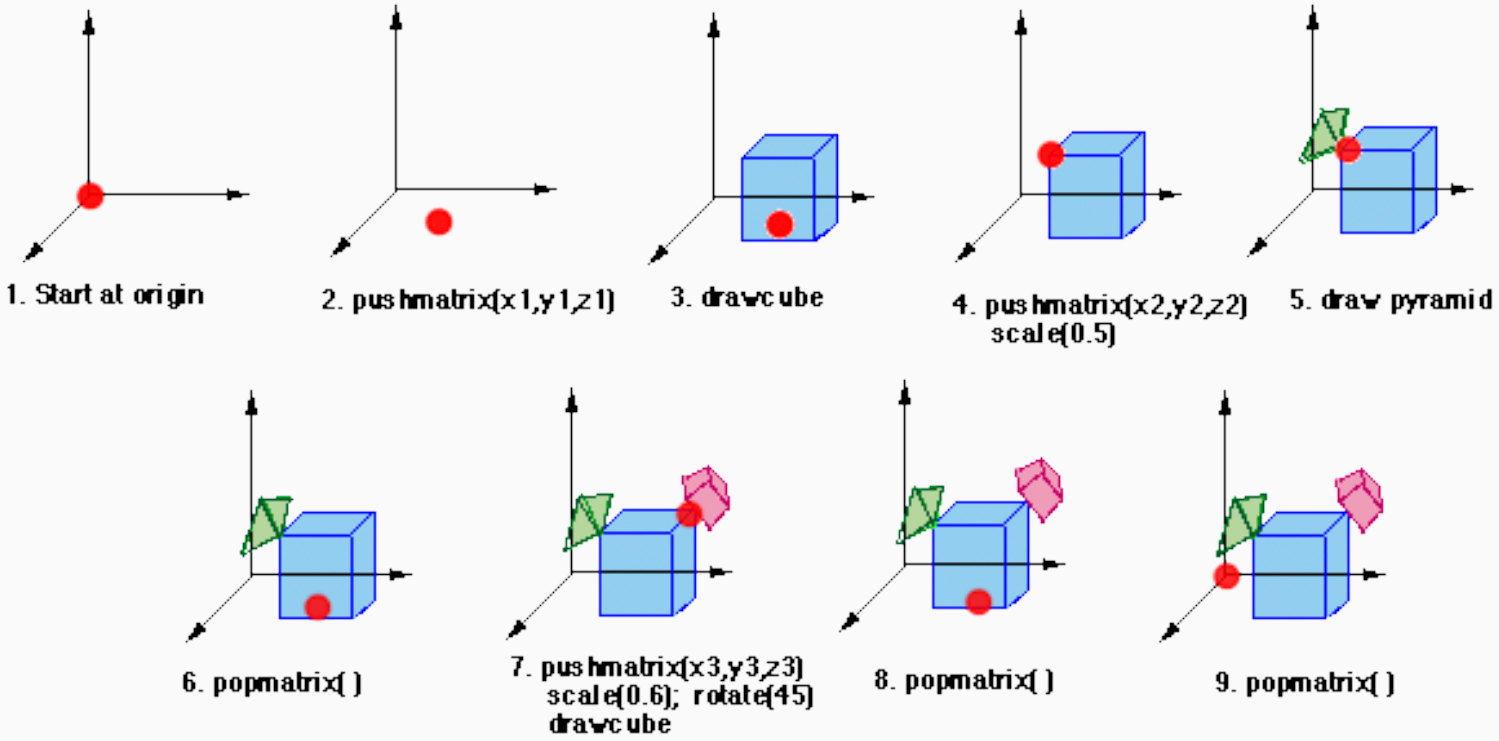
1. Translate
2. Rotate

Transformation Homogenous Coordinates

$$\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{11}(\theta) & R_{12}(\theta) & R_{13}(\theta) & 0 \\ R_{21}(\theta) & R_{22}(\theta) & R_{23}(\theta) & 0 \\ R_{31}(\theta) & R_{32}(\theta) & R_{33}(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotate θ degrees around x-axis at $[a, b, c]^{\mathrm{T}}$
- Remember that matrix multiplications are not cummutative

Transformation Stack

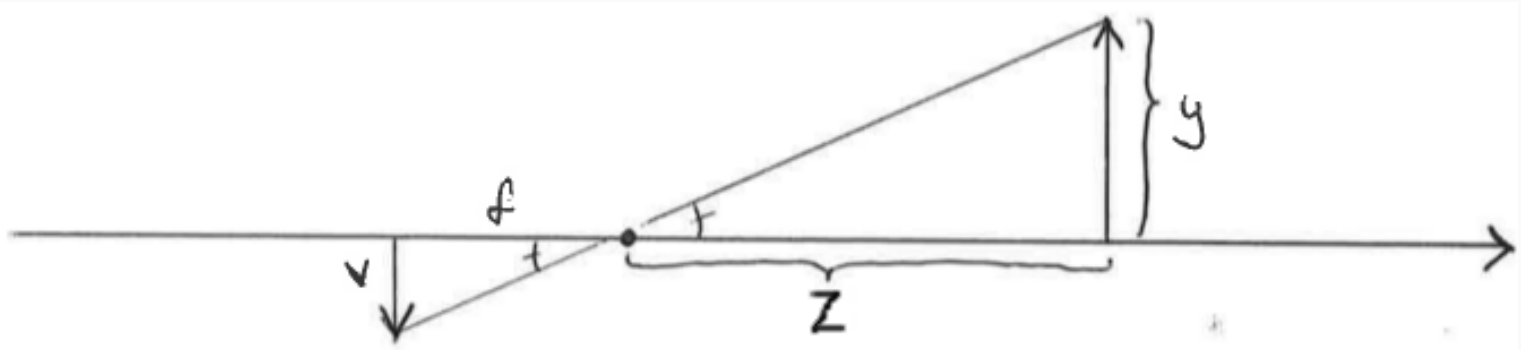


Raytracing

Camera

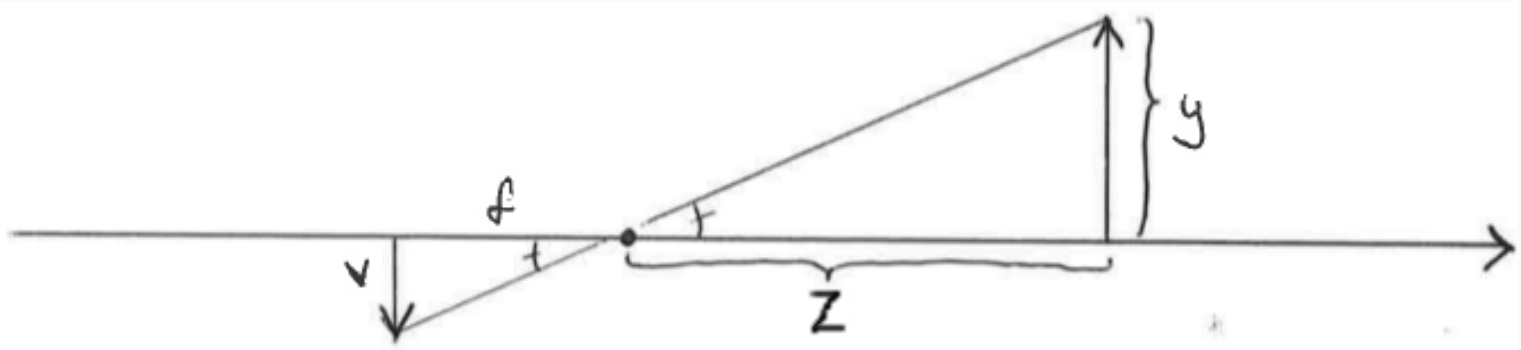
- Cameras create images
- Lens
  - captures rays
- Focal lenth
- Shutter opening

Pinhole camera



- Simplest form camera, where one ray of light lits up each pixel
- Defined by a *location*, *focal length*, *view-direction* and *up-direction*
- is this a realistic camera?

Pinhole camera

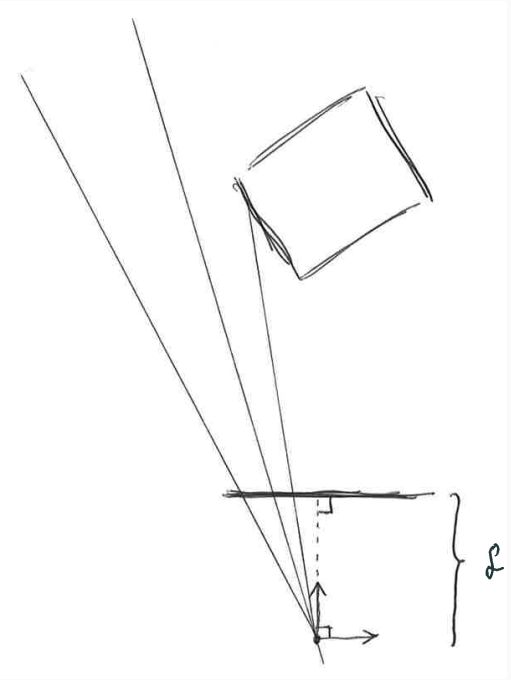


$$\frac{v}{f} = \frac{y}{z}$$

$$\Rightarrow v = \frac{f}{z} y$$

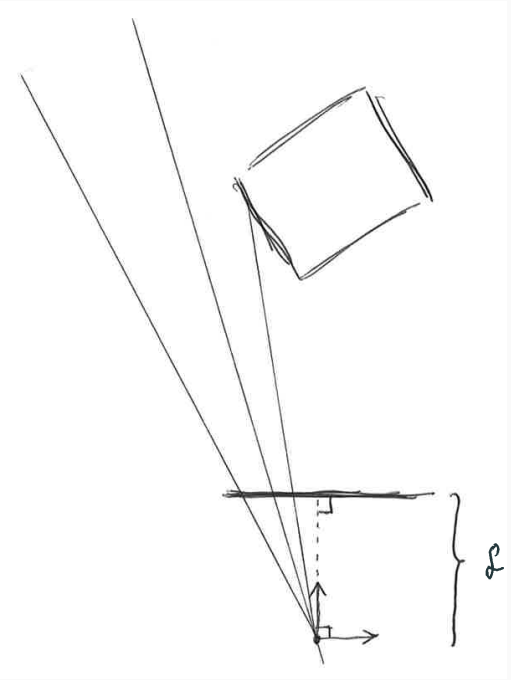
Raycasting

1. Camera at $[0, 0, 0]^{\mathrm{T}}$



Raycasting

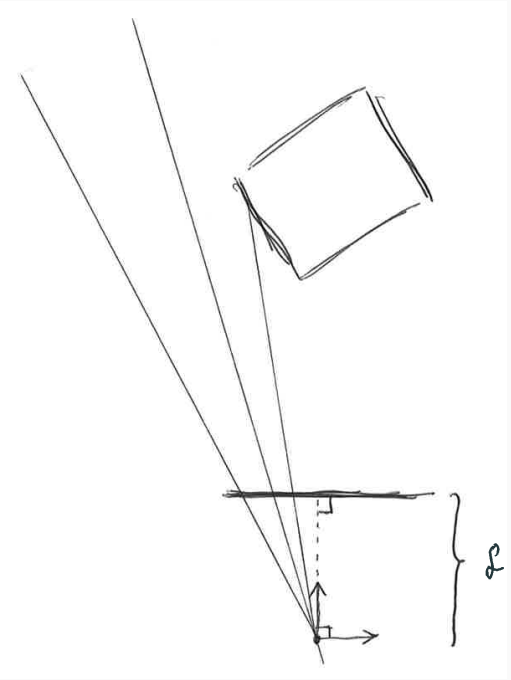
1. Camera at $[0, 0, 0]^{\mathrm{T}}$
2. View direction along z-axis $[0, 0, 1]^{\mathrm{T}}$



Raycasting

1. Camera at $[0, 0, 0]^{\mathrm{T}}$
2. View direction along z-axis $[0, 0, 1]^{\mathrm{T}}$
3. Image plane

$$\left\{ [u, v, f]^{\mathrm{T}} \,||\, -\frac{W}{2} \leq u < \frac{W}{2}, -\frac{H}{2} < v < \frac{H}{2}, f \right.$$



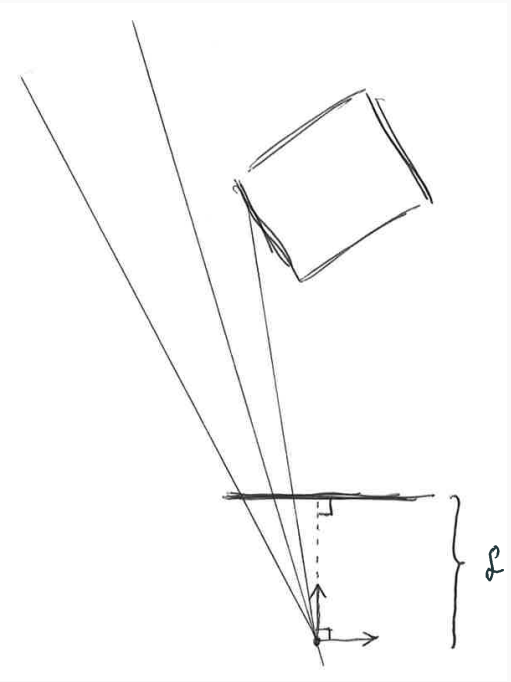
Raycasting

1. Camera at $[0, 0, 0]^{\mathrm{T}}$
2. View direction along z-axis $[0, 0, 1]^{\mathrm{T}}$
3. Image plane

$$\left\{ [u, v, f]^{\mathrm{T}} || -\frac{W}{2} \leq u < \frac{W}{2}, -\frac{H}{2} < v < \frac{H}{2}, f \right.$$

4. Compute normalised vector from origin to image-plane

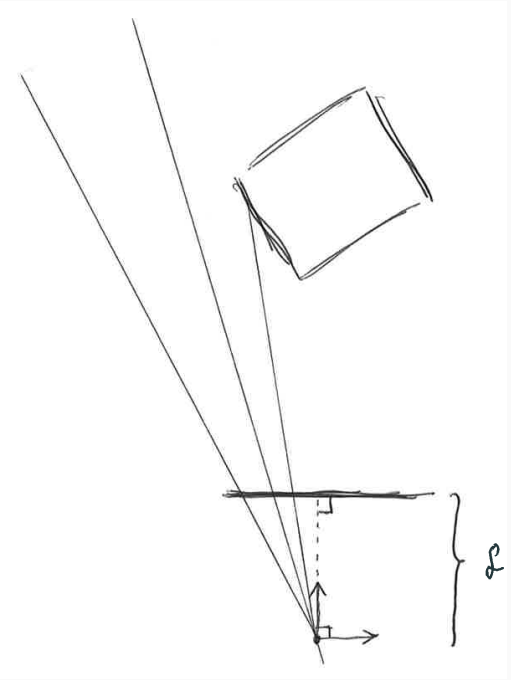
$$\mathbf{d}_{ij} = \frac{1}{\sqrt{[u_i, v_j, f][u_i, v_j, f]^{\mathrm{T}}}} [u_i, v_j, f]^{\mathrm{T}}$$



Raycasting

1. Camera at $[0, 0, 0]^{\mathrm{T}}$
2. View direction along z-axis $[0, 0, 1]^{\mathrm{T}}$
3. Image plane

$$\left\{ [u, v, f]^{\mathrm{T}} || -\frac{W}{2} \leq u < \frac{W}{2}, -\frac{H}{2} < v < \frac{H}{2}, f \right.$$



4. Compute normalised vector from origin to image-plane

$$\mathbf{d}_{ij} = \frac{1}{\sqrt{[u_i, v_j, f][u_i, v_j, f]^{\mathrm{T}}}} [u_i, v_j, f]^{\mathrm{T}}$$

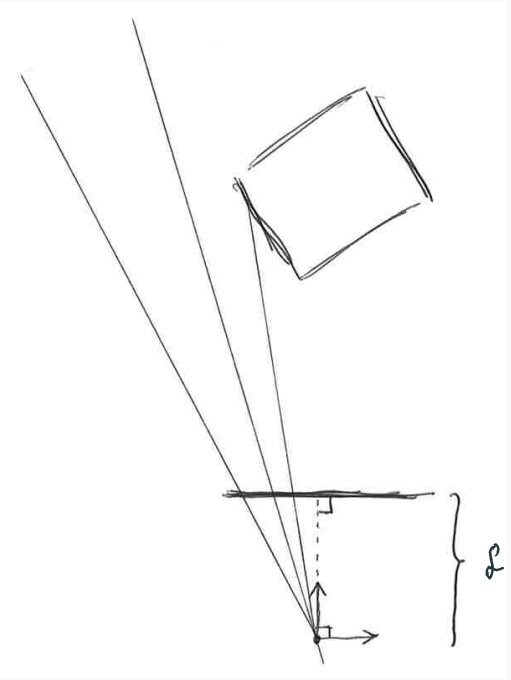
5. Parmatrise Ray

$$\mathbf{r}_{ij} = \mathbf{s} + t \cdot \mathbf{d}_{ij}$$

Raycasting

1. Camera at $[0, 0, 0]^{\mathrm{T}}$
2. View direction along z-axis $[0, 0, 1]^{\mathrm{T}}$
3. Image plane

$$\left\{ [u, v, f]^{\mathrm{T}} || -\frac{W}{2} \leq u < \frac{W}{2}, -\frac{H}{2} < v < \frac{H}{2}, f \right.$$



4. Compute normalised vector from origin to image-plane

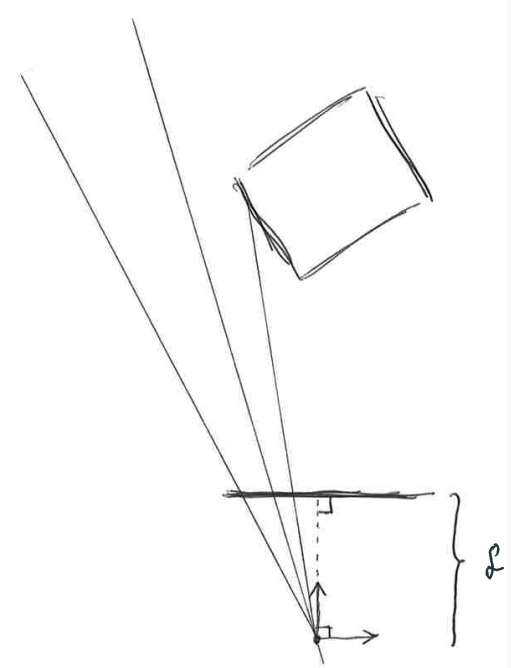
$$\mathbf{d}_{ij} = \frac{1}{\sqrt{[u_i, v_j, f][u_i, v_j, f]^{\mathrm{T}}}} [u_i, v_j, f]^{\mathrm{T}}$$

5. Parmatrise Ray

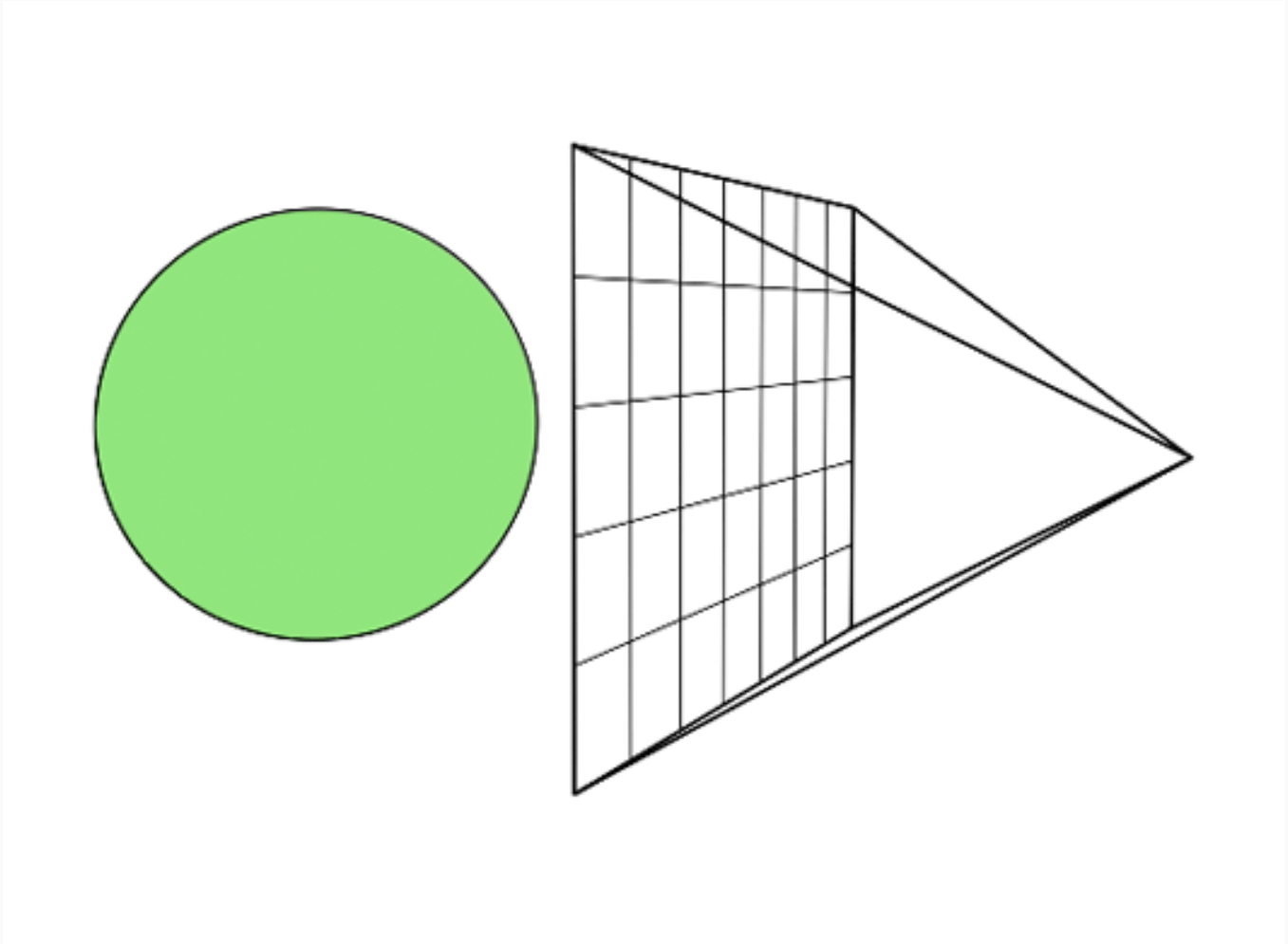
$$\mathbf{r}_{ij} = \mathbf{s} + t \cdot \mathbf{d}_{ij}$$

Raycasting

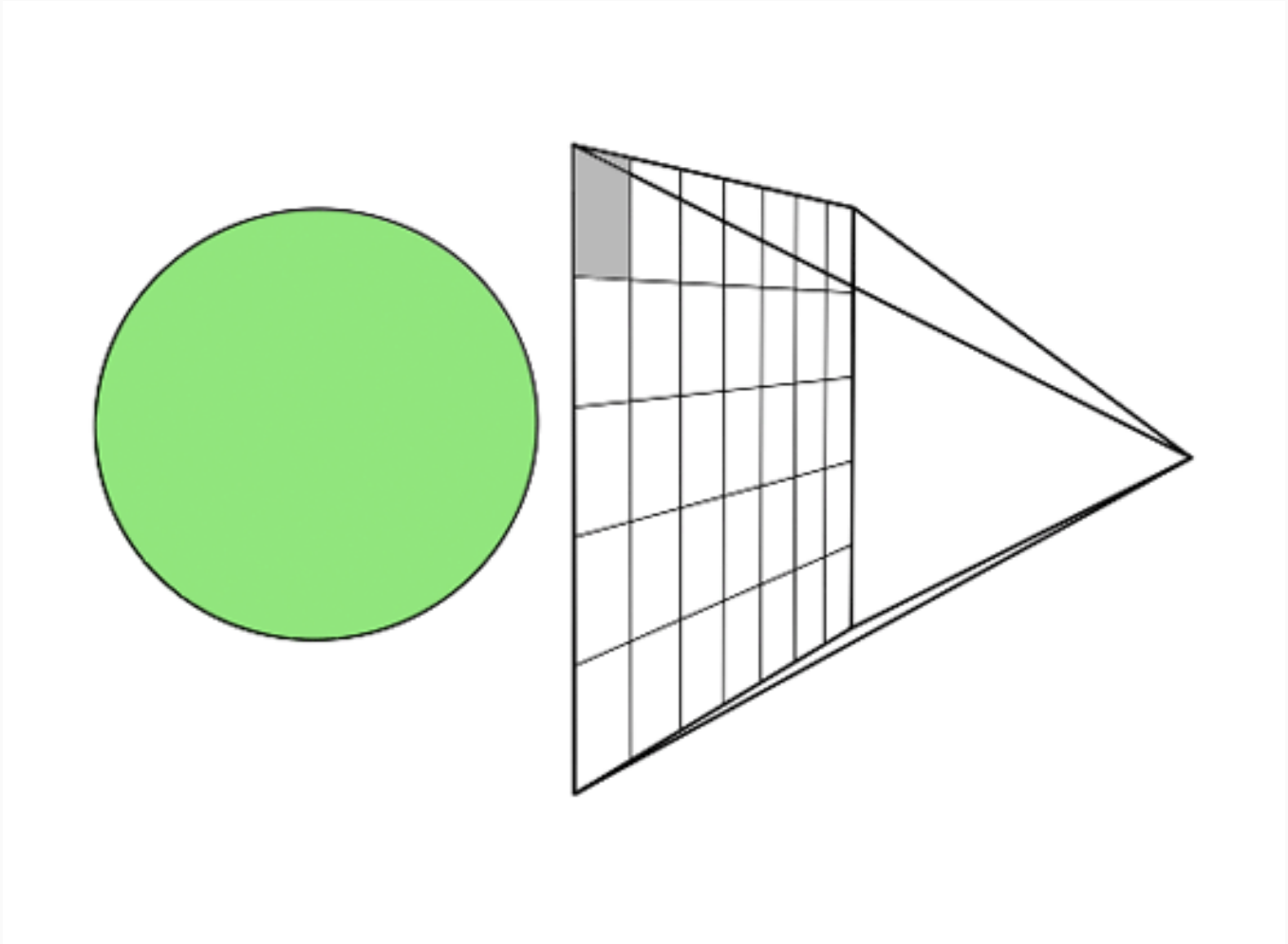
- Trace a ray for each pixel
- Compute intersection with all objects in world
- Plot colour of intersecting surface
- No "lighting" often called Raycaster



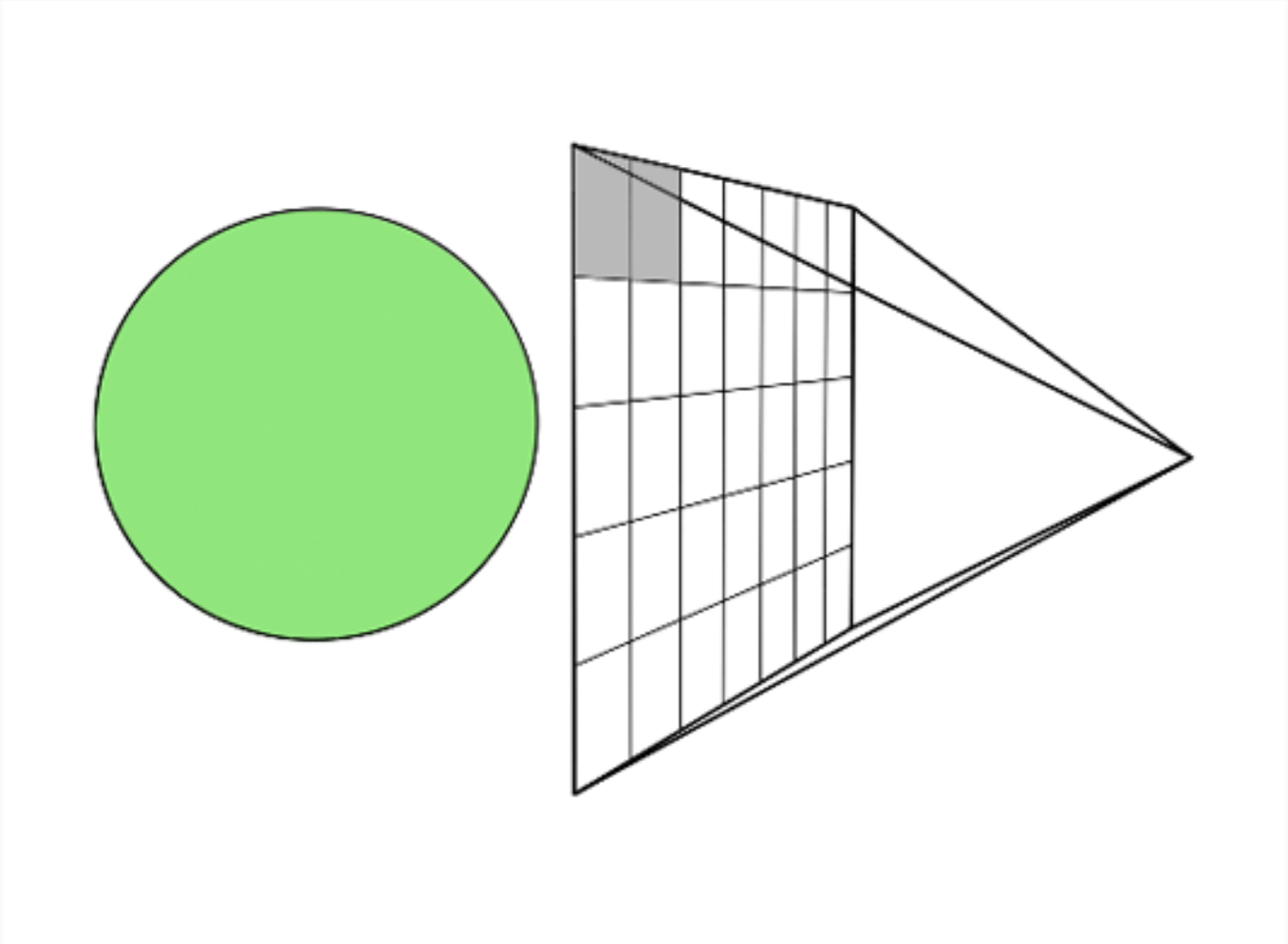
Raycasting Anim



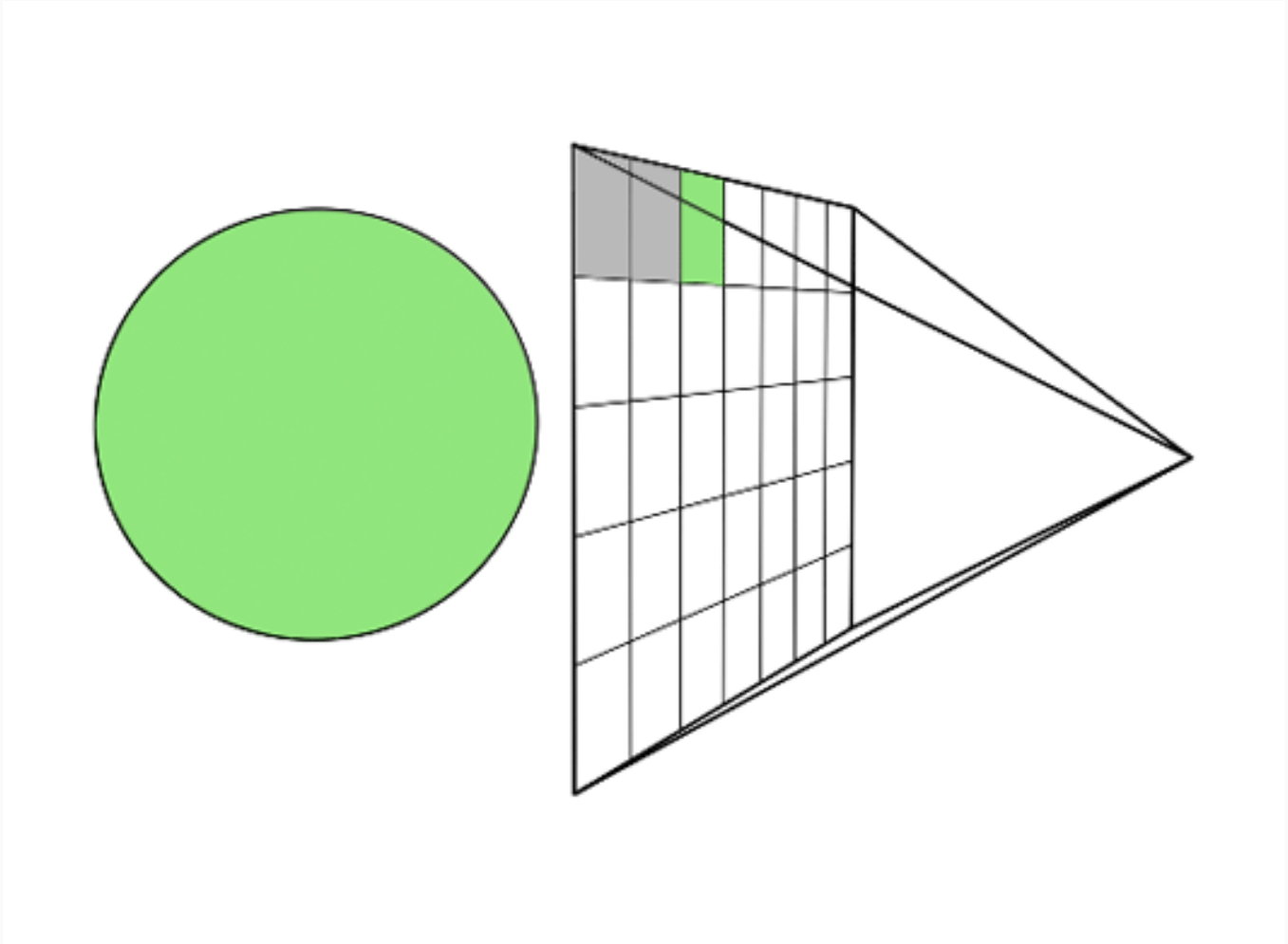
Raycasting Anim



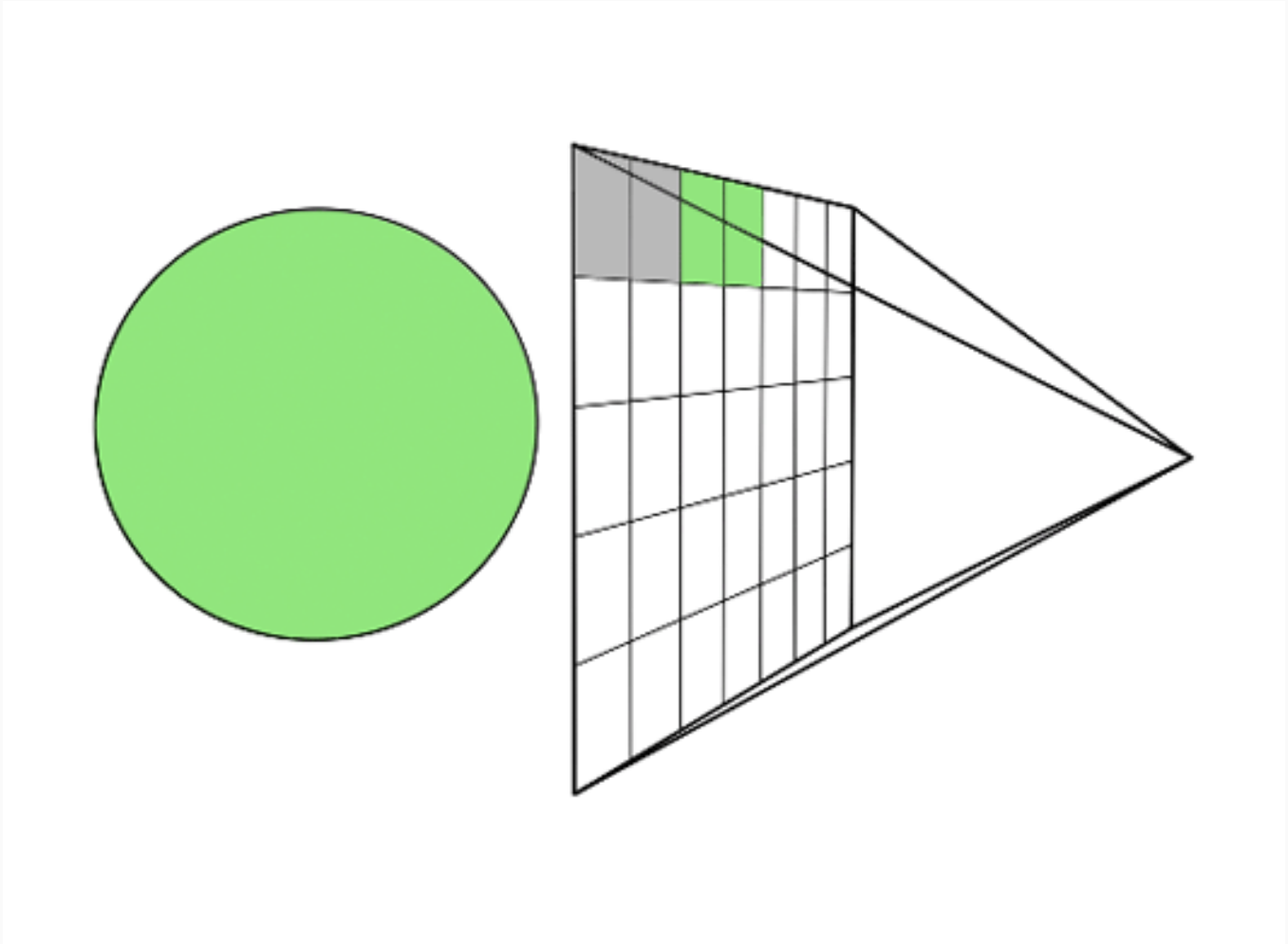
Raycasting Anim



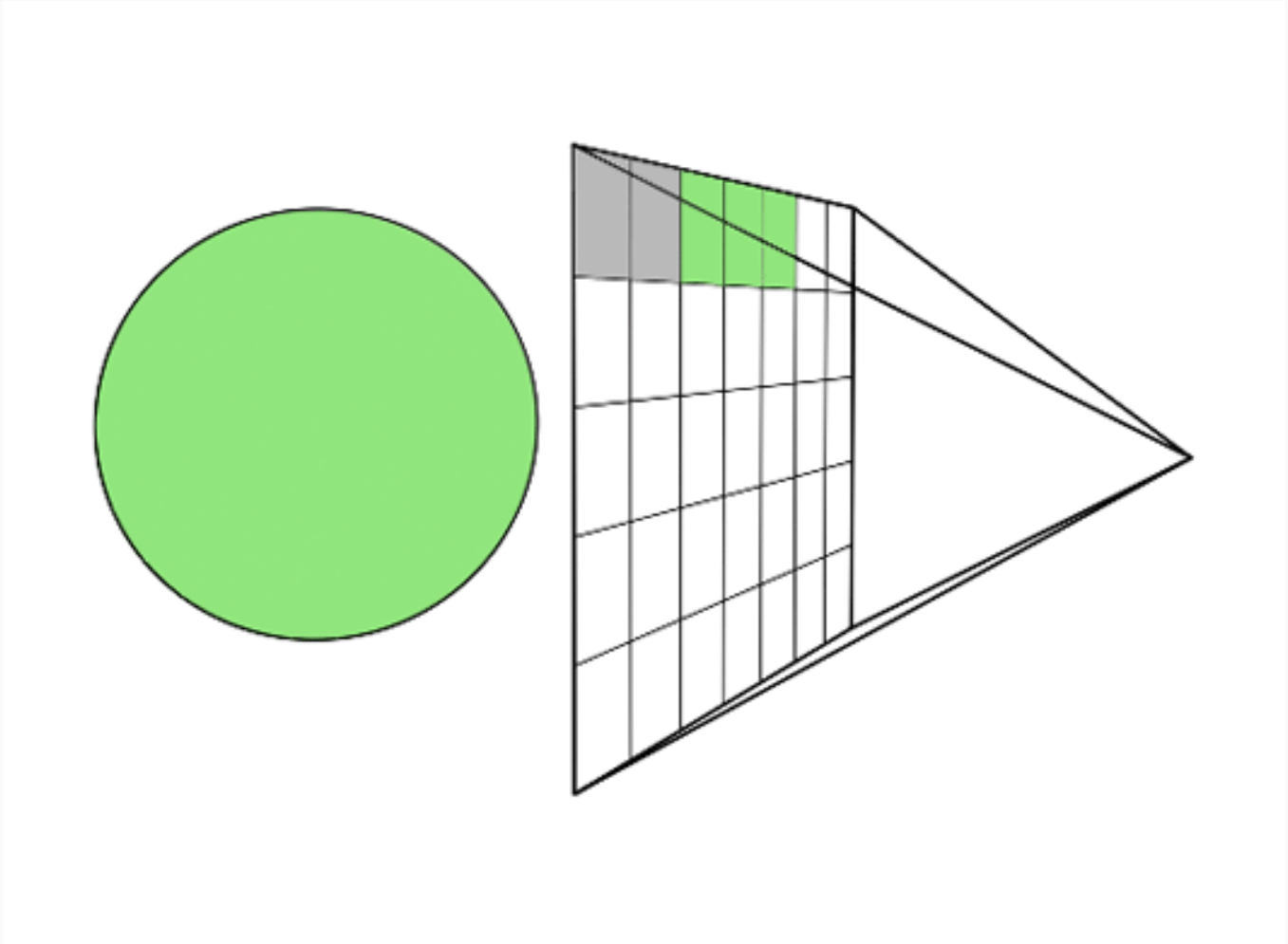
Raycasting Anim



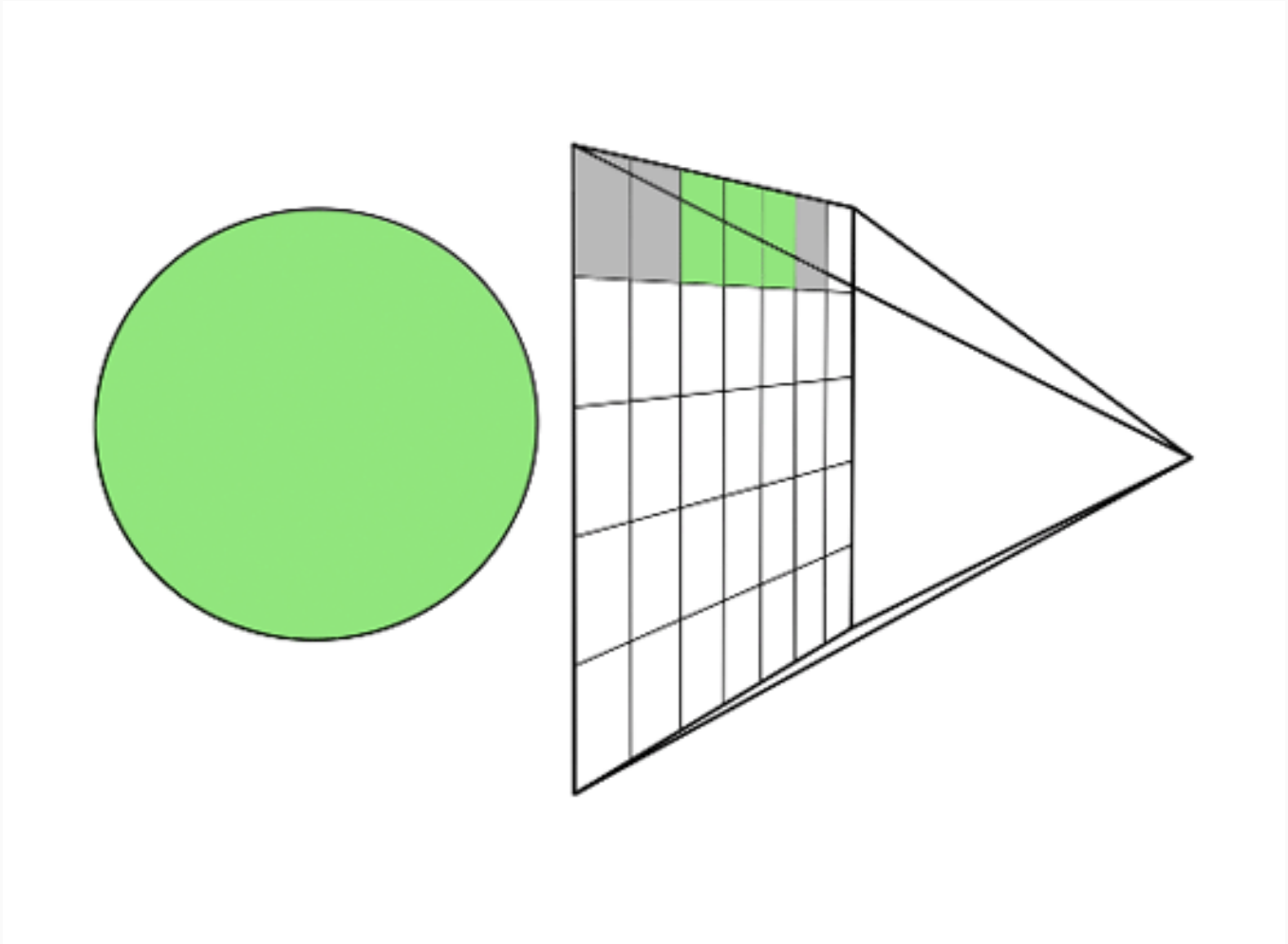
Raycasting Anim



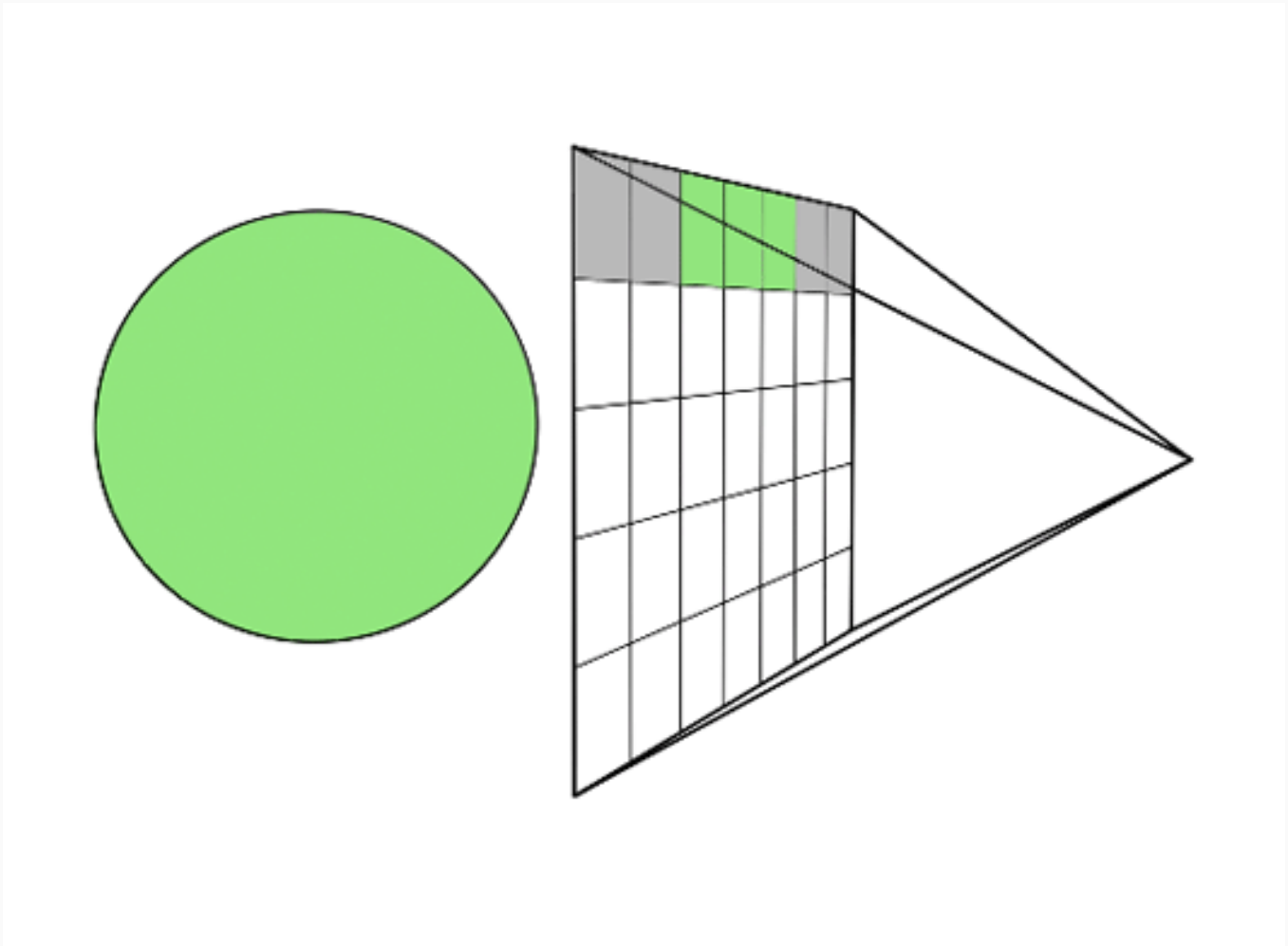
Raycasting Anim



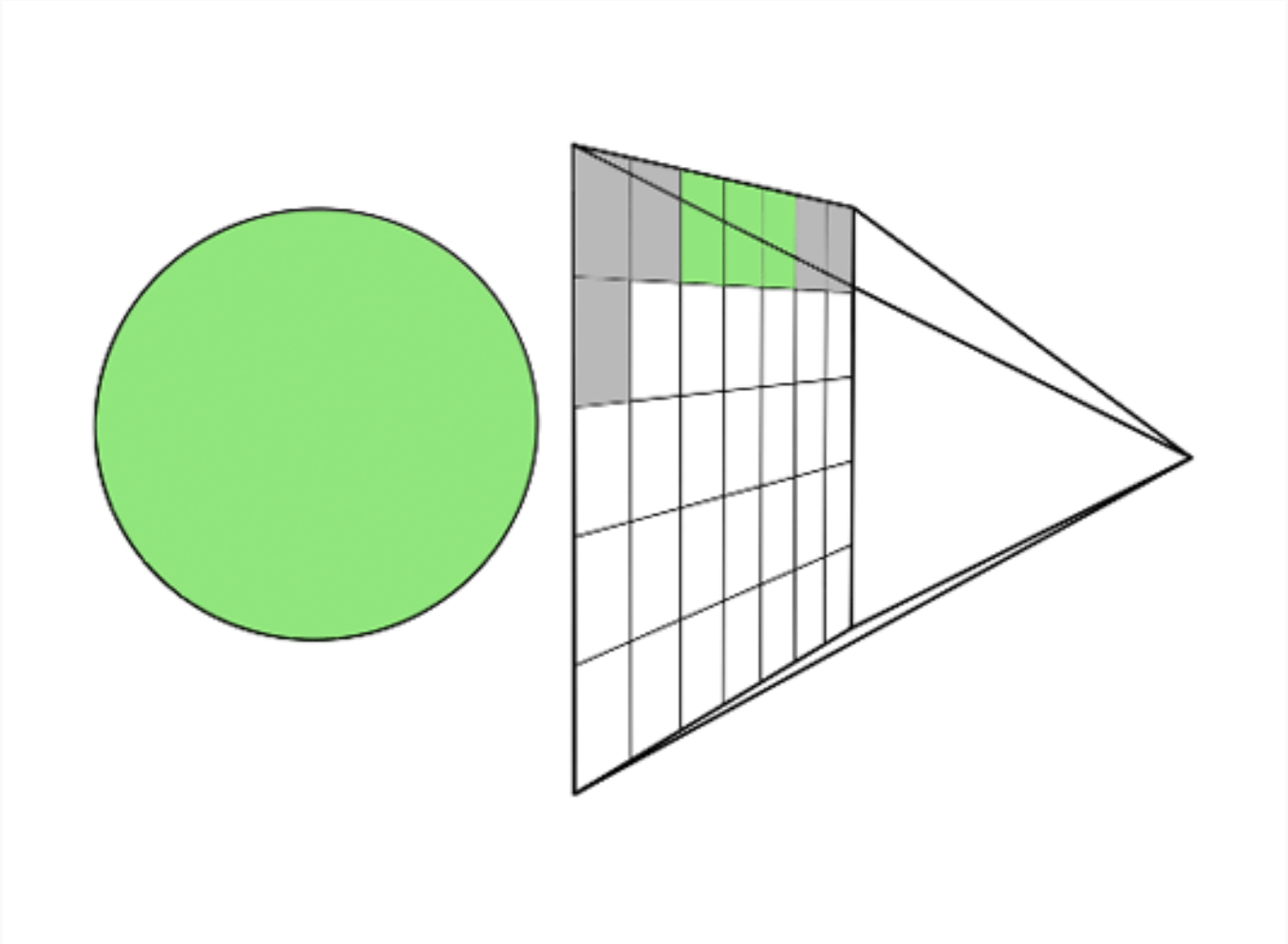
Raycasting Anim



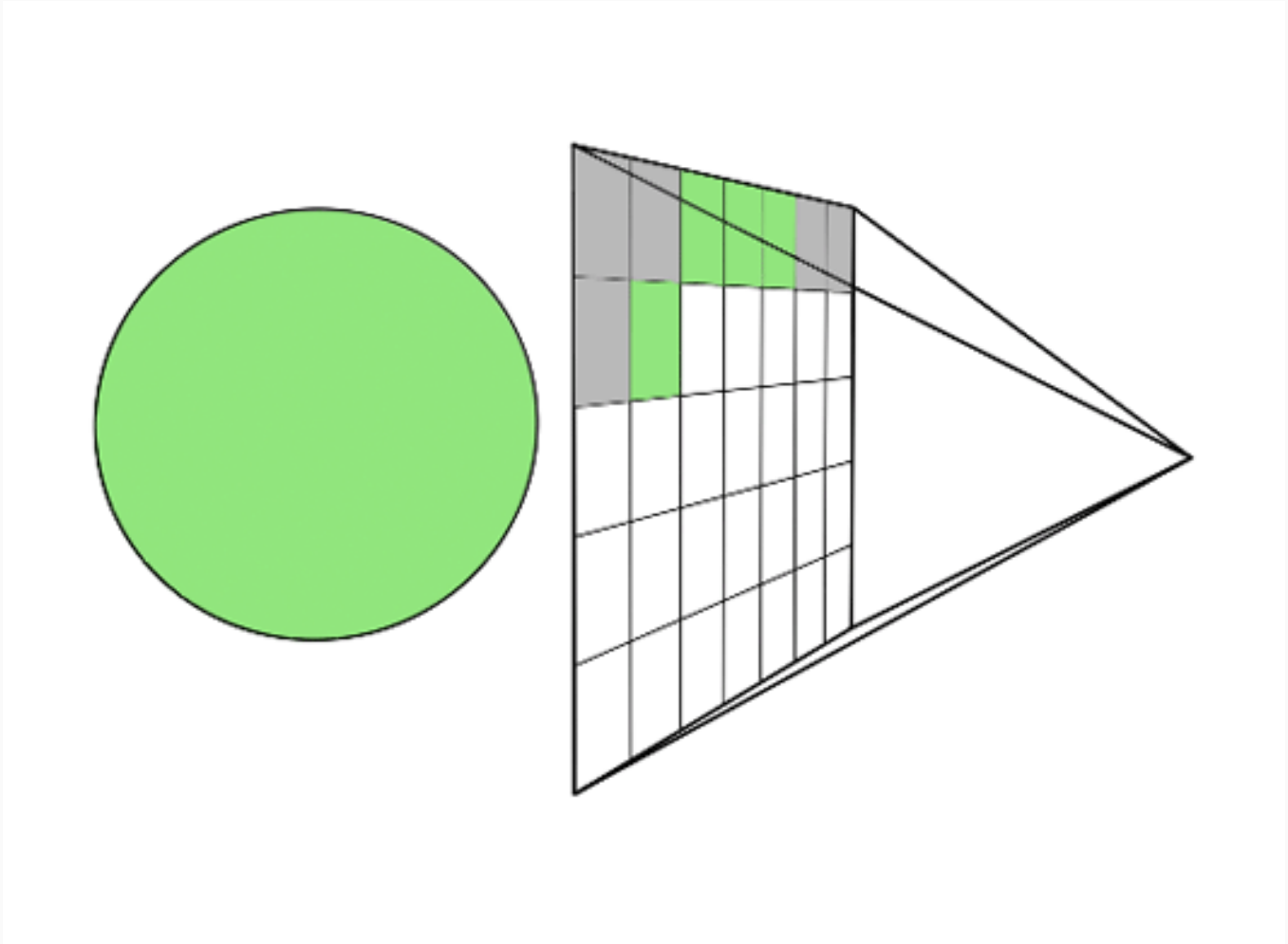
Raycasting Anim



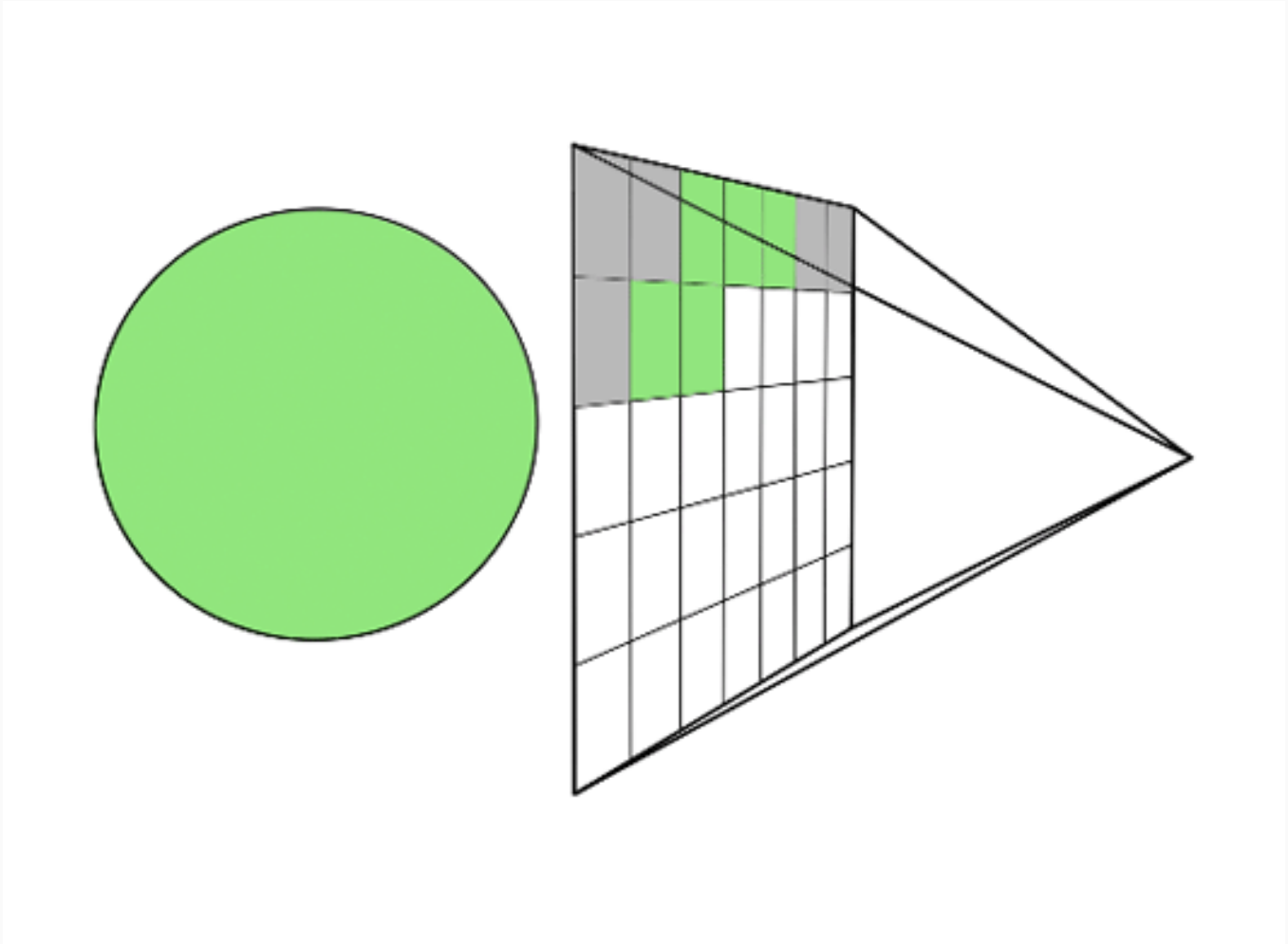
Raycasting Anim



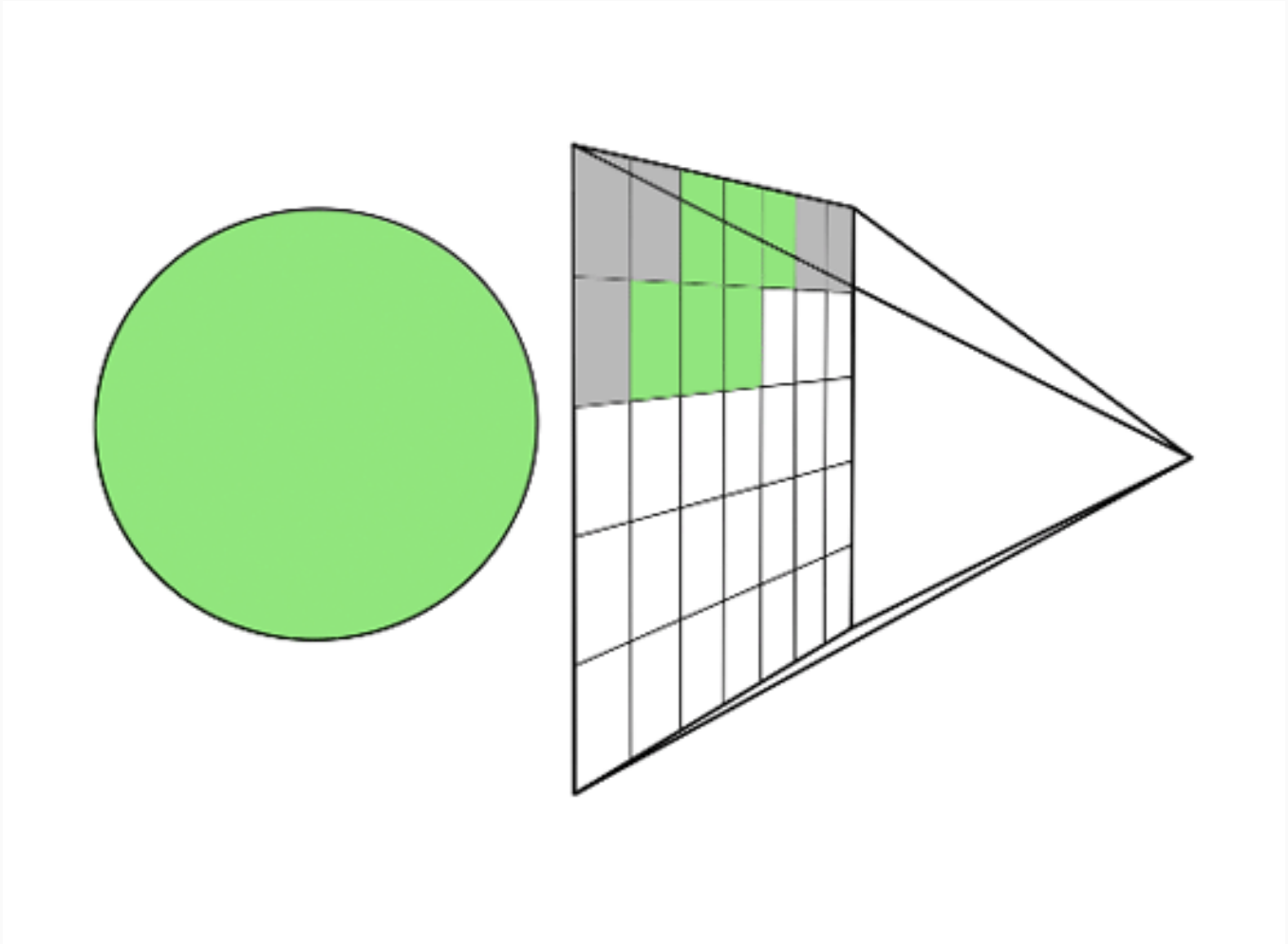
Raycasting Anim



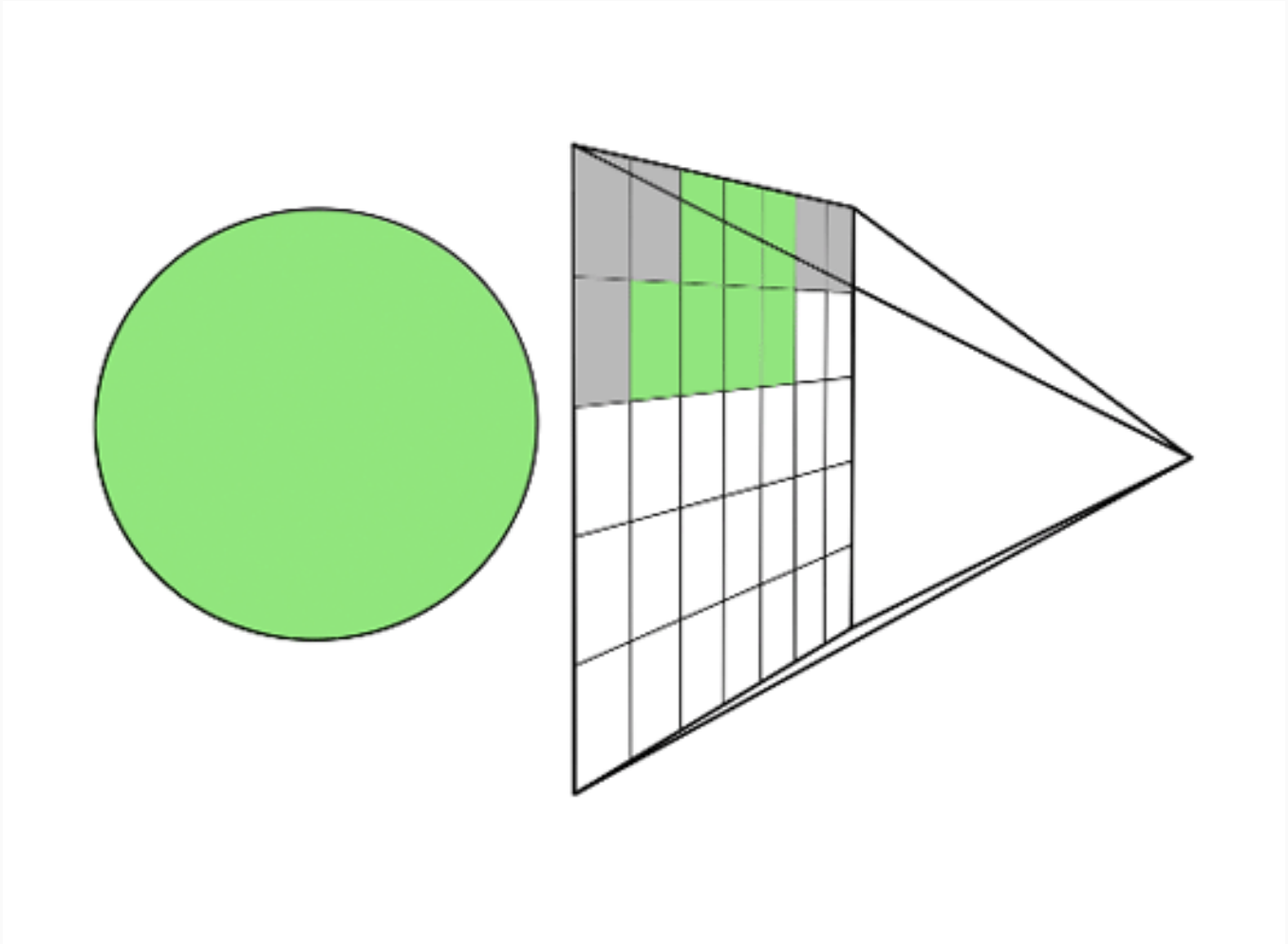
Raycasting Anim



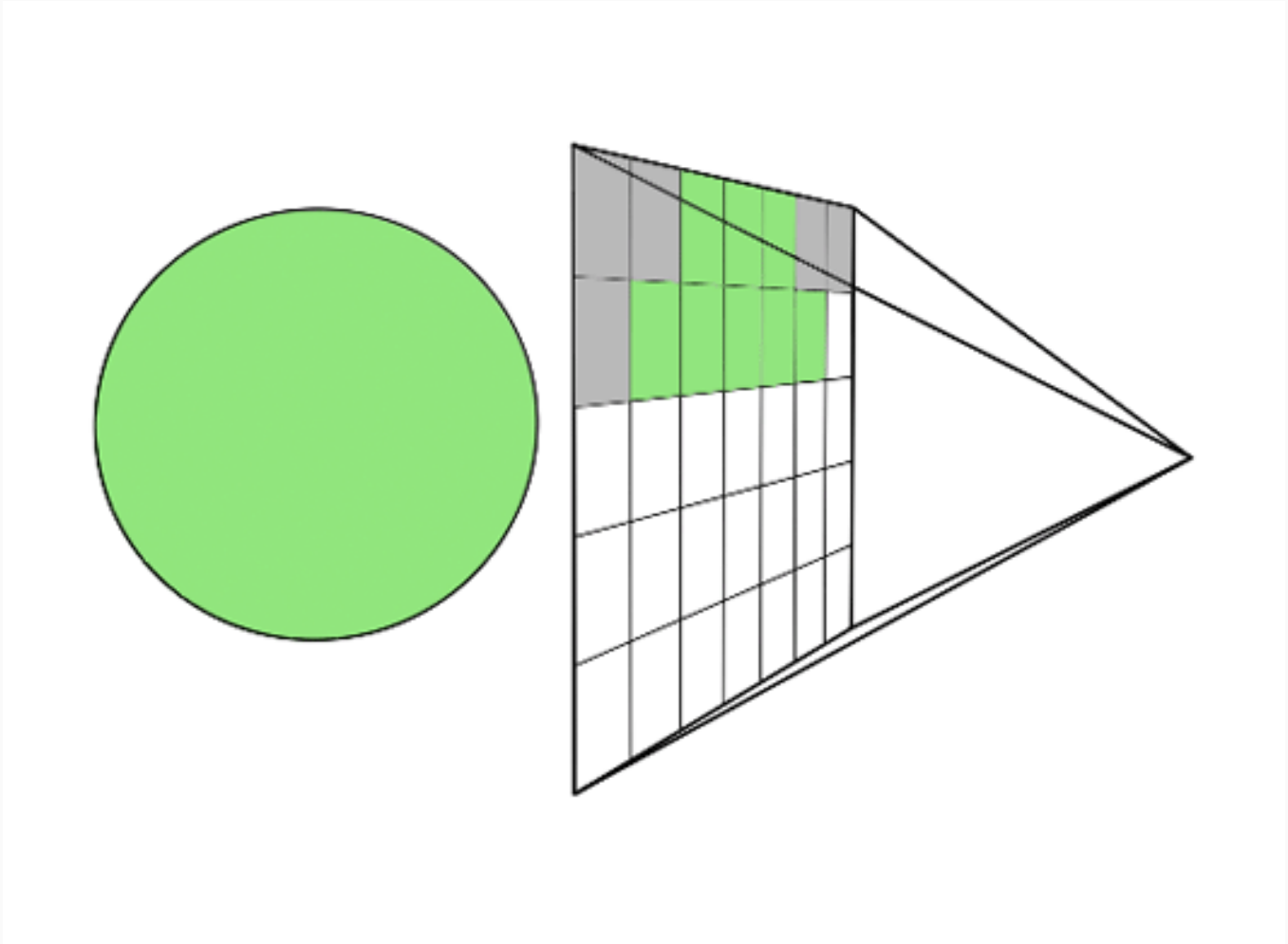
Raycasting Anim



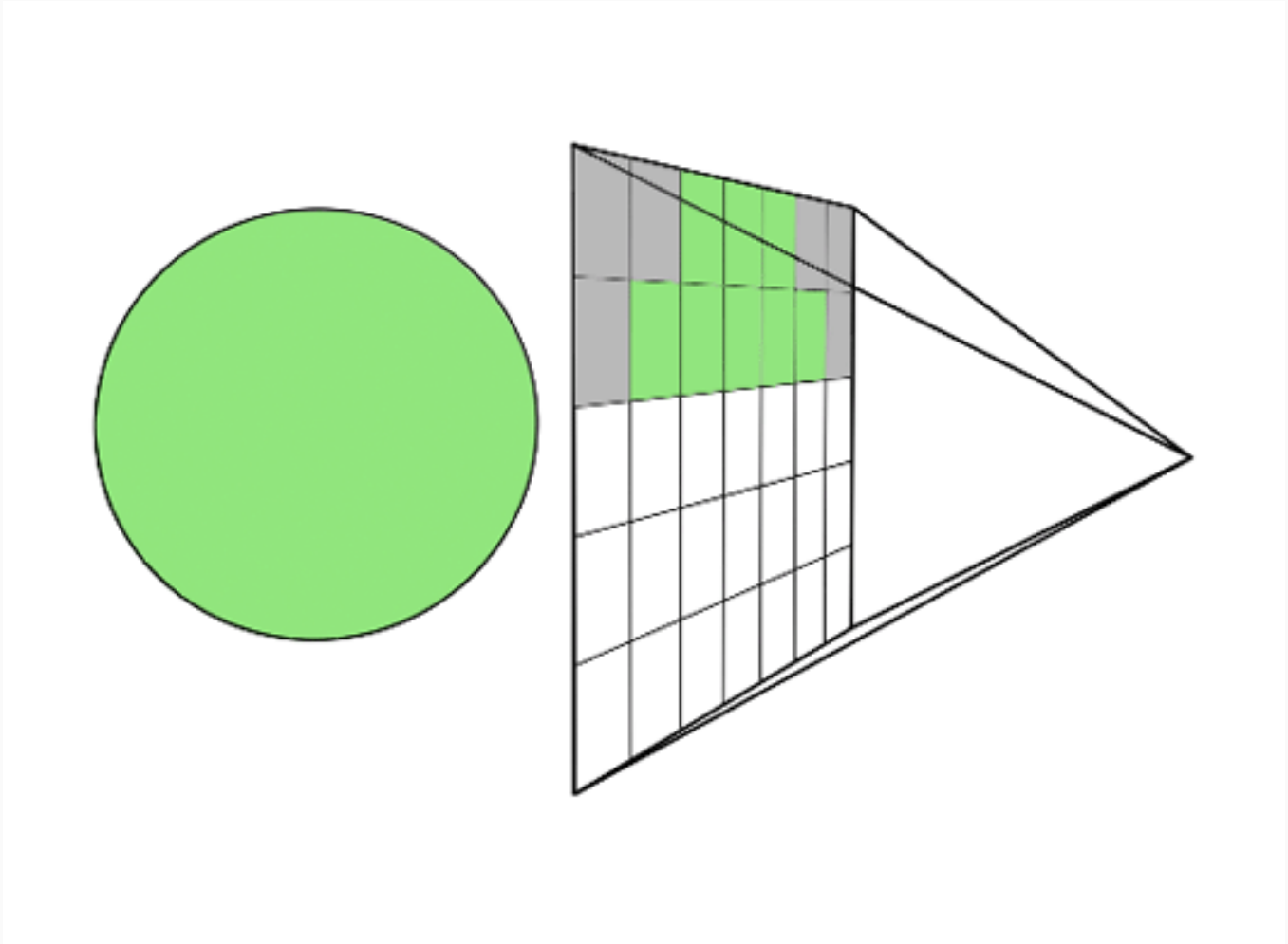
Raycasting Anim



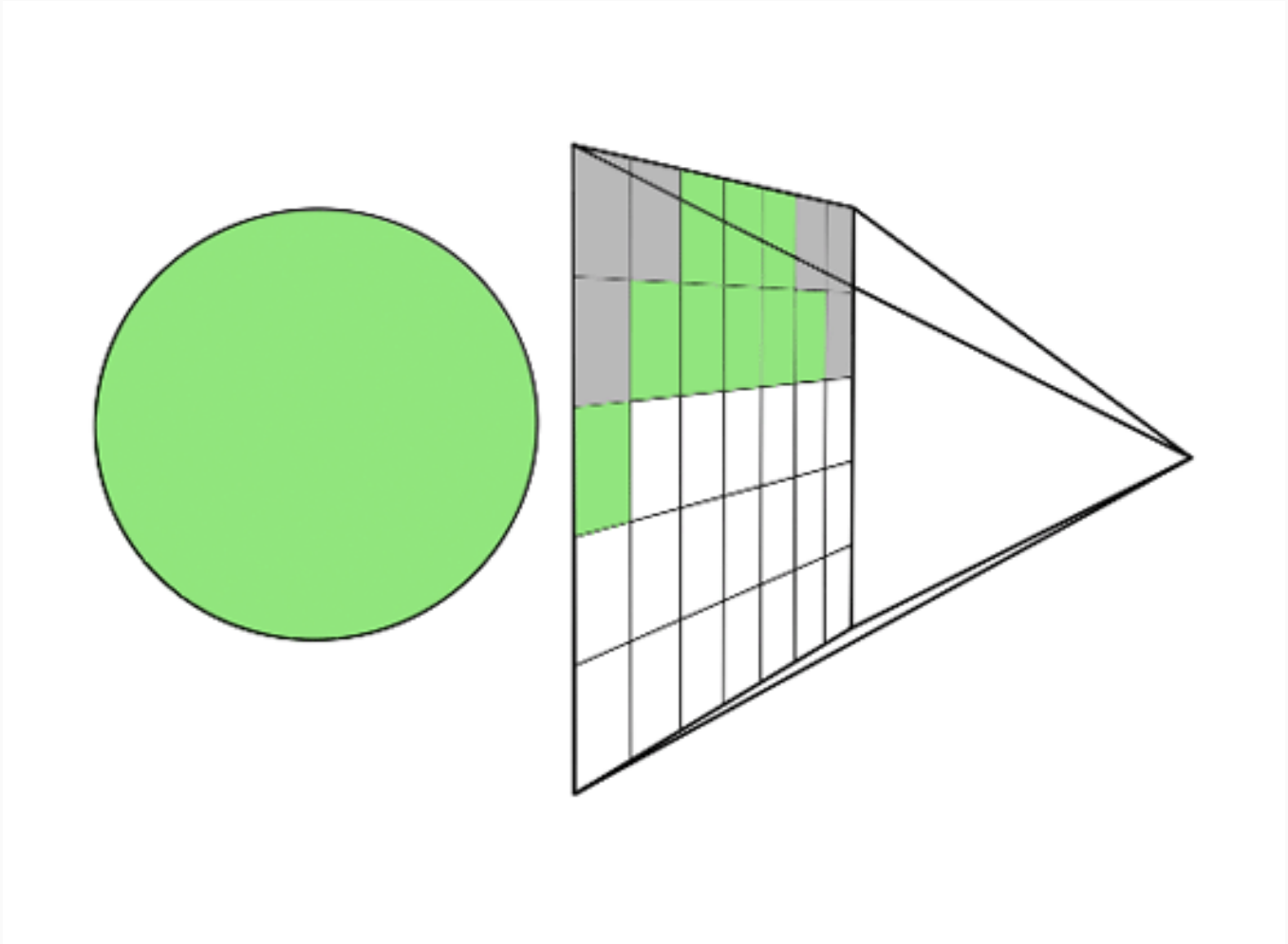
Raycasting Anim



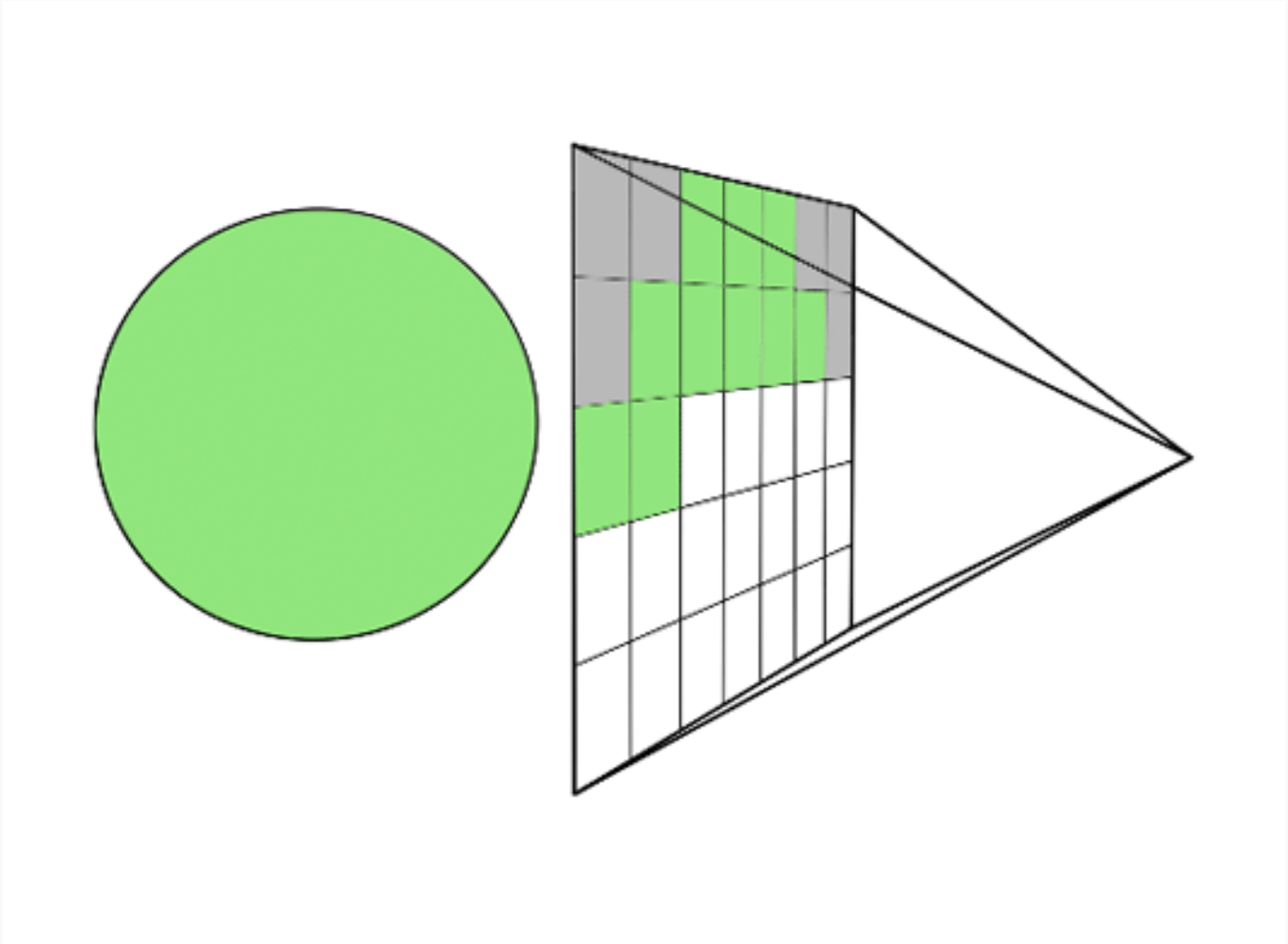
Raycasting Anim



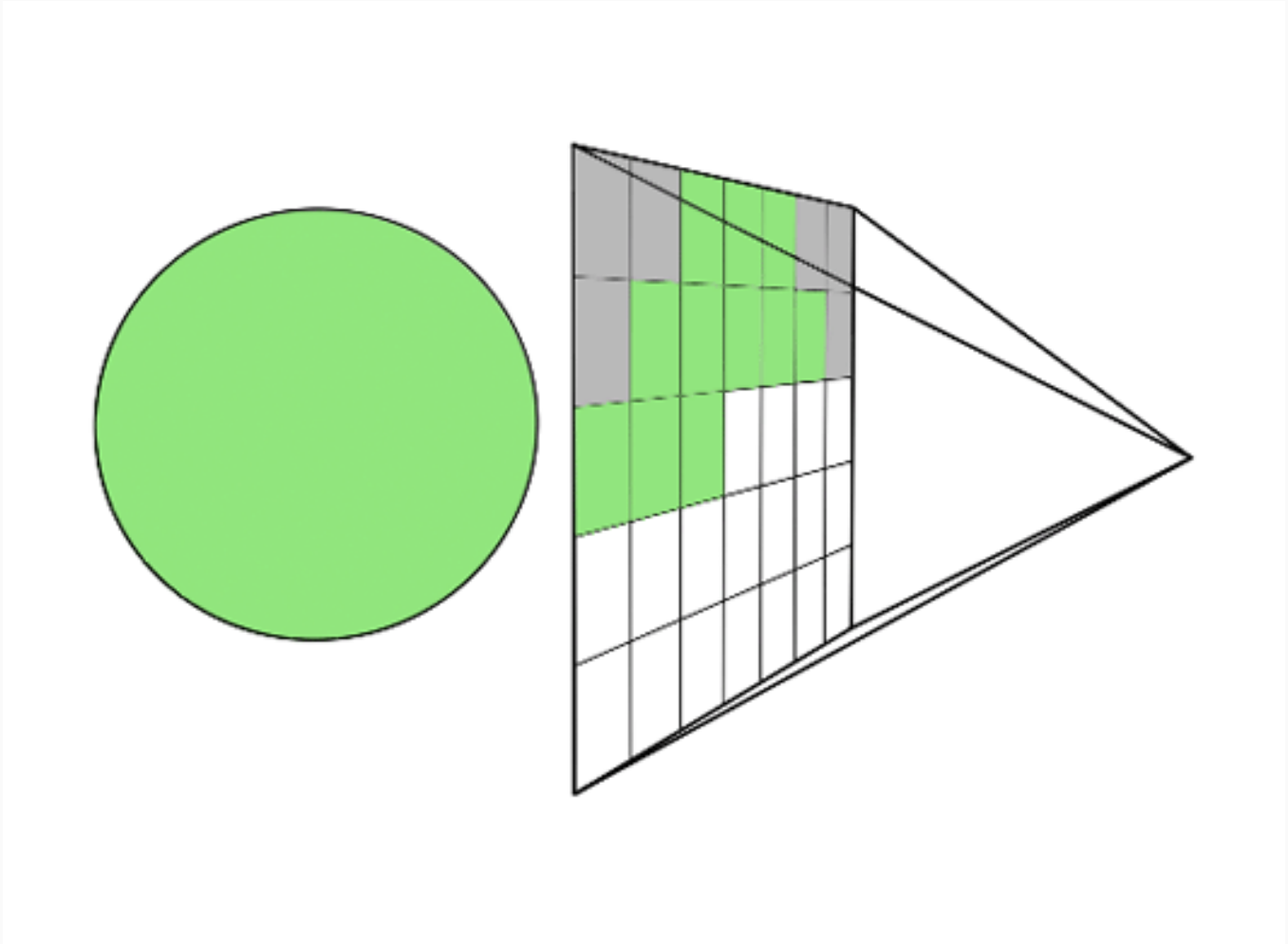
Raycasting Anim



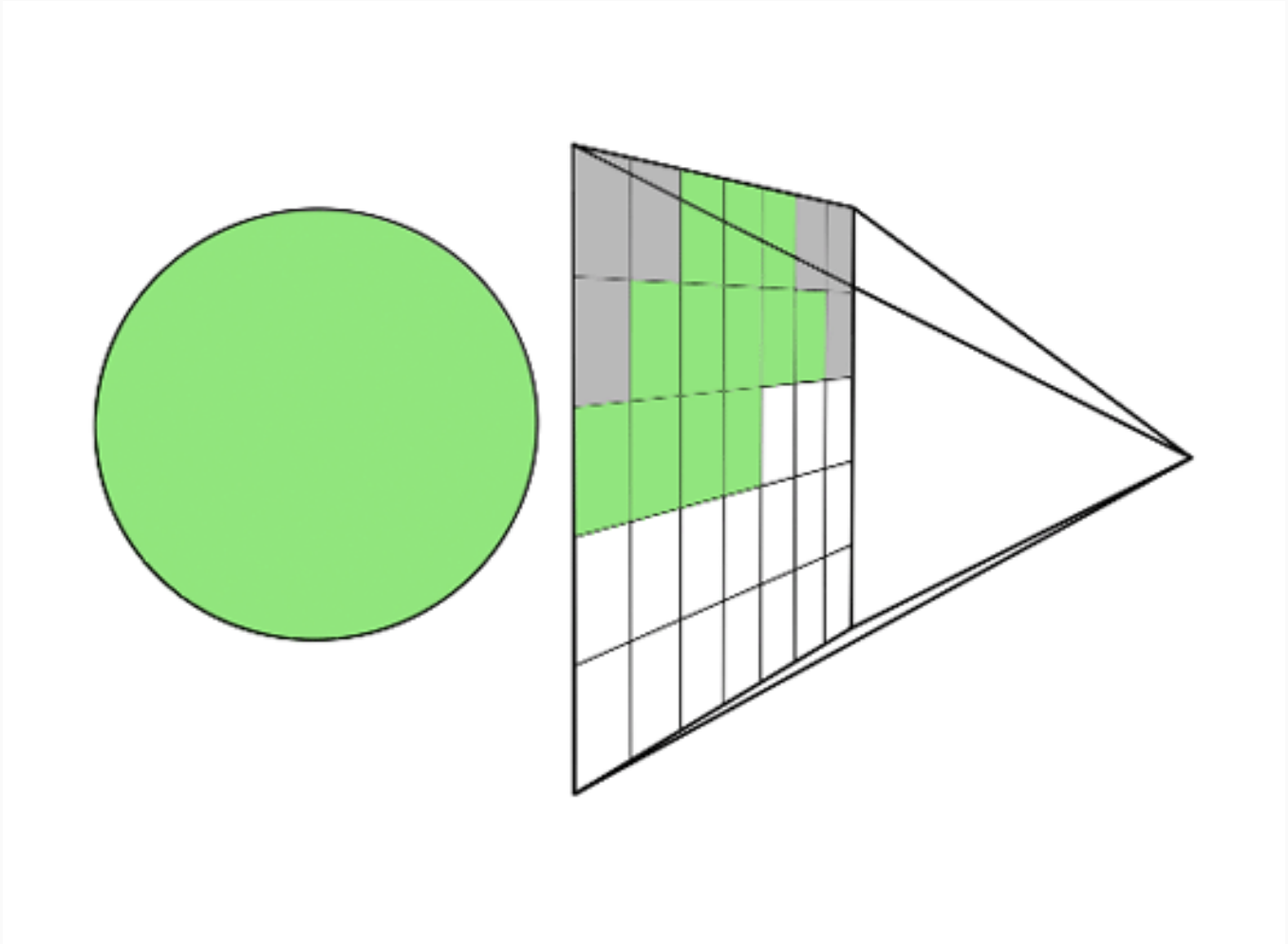
Raycasting Anim



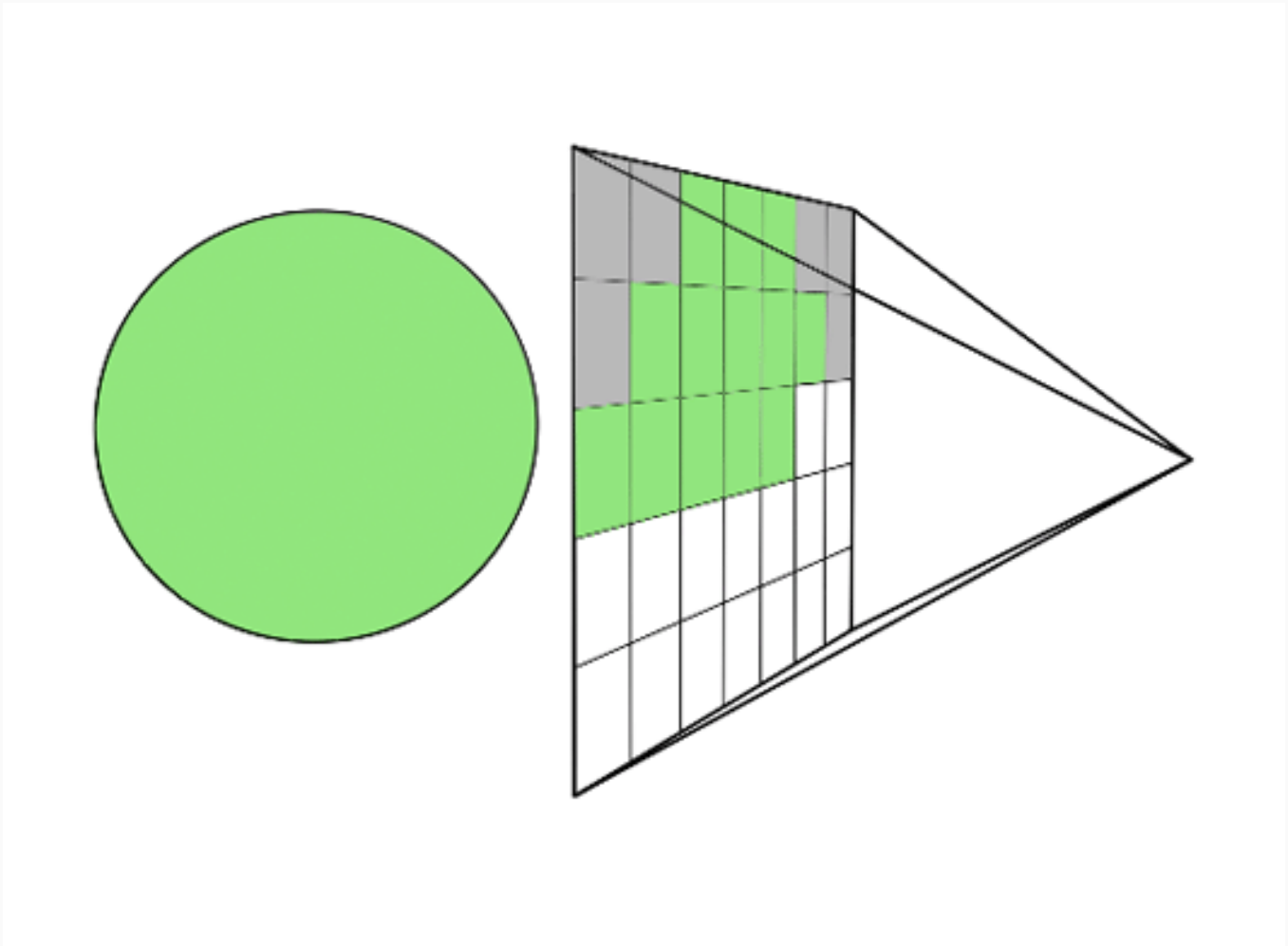
Raycasting Anim



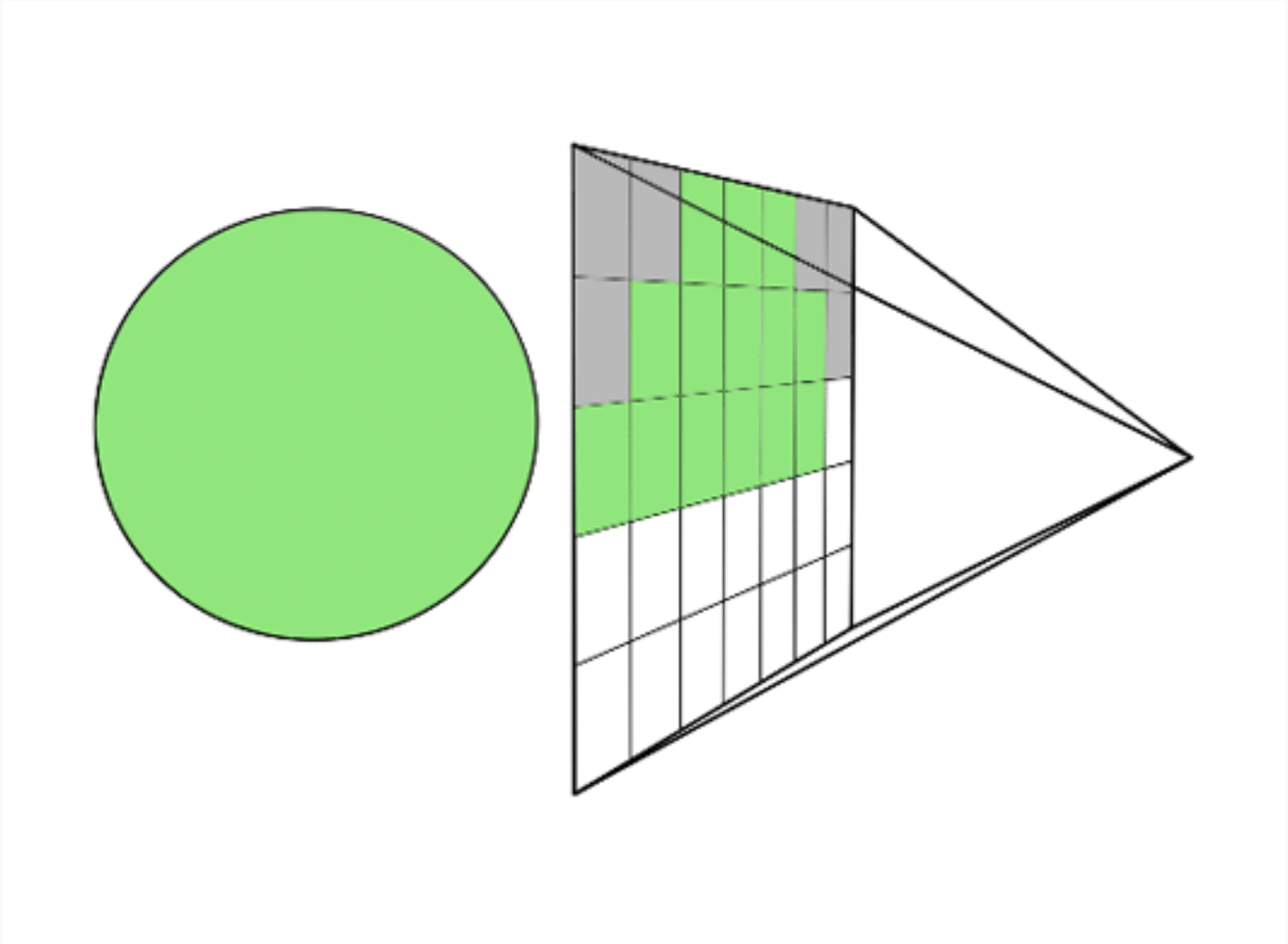
Raycasting Anim



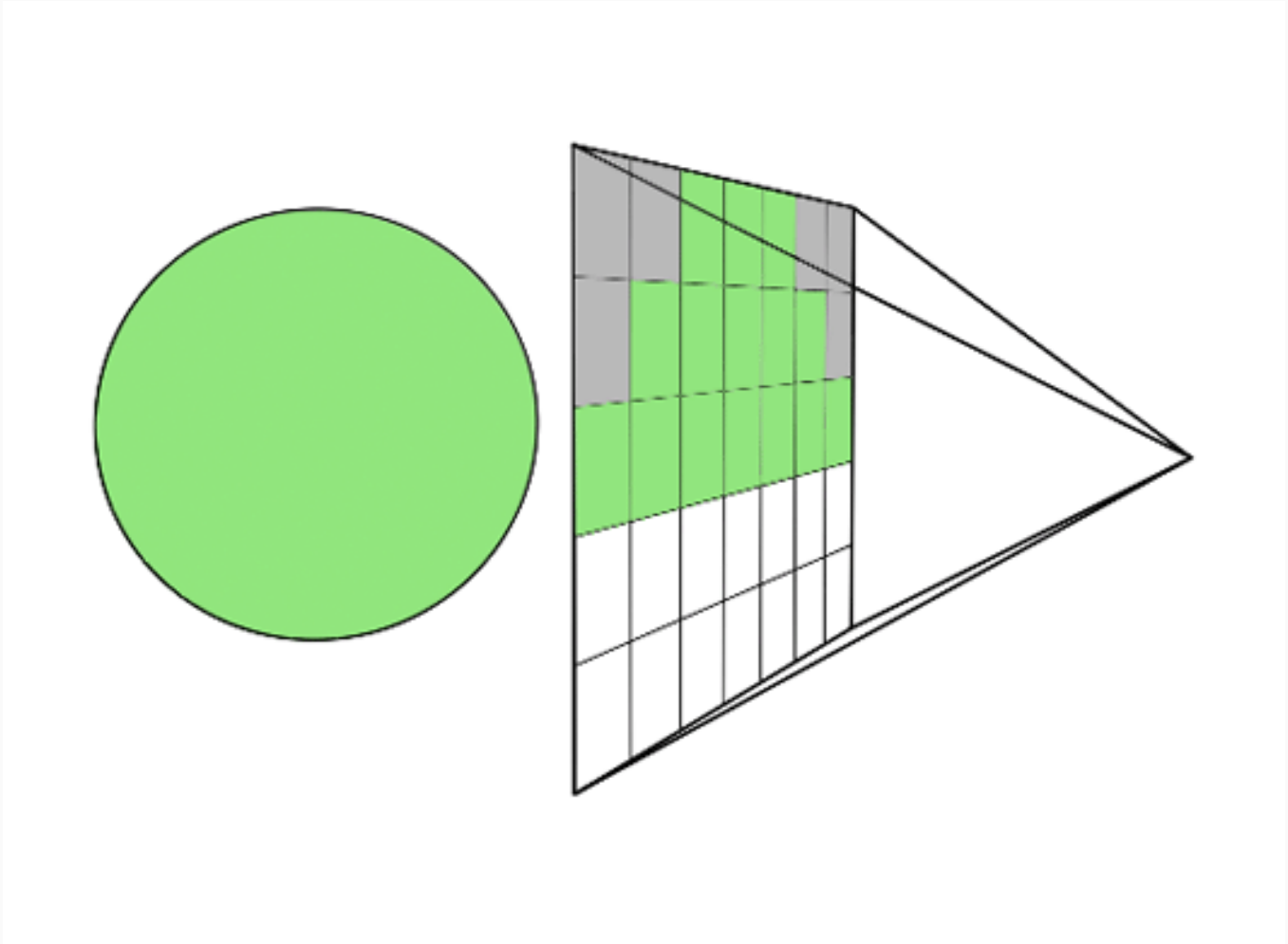
Raycasting Anim



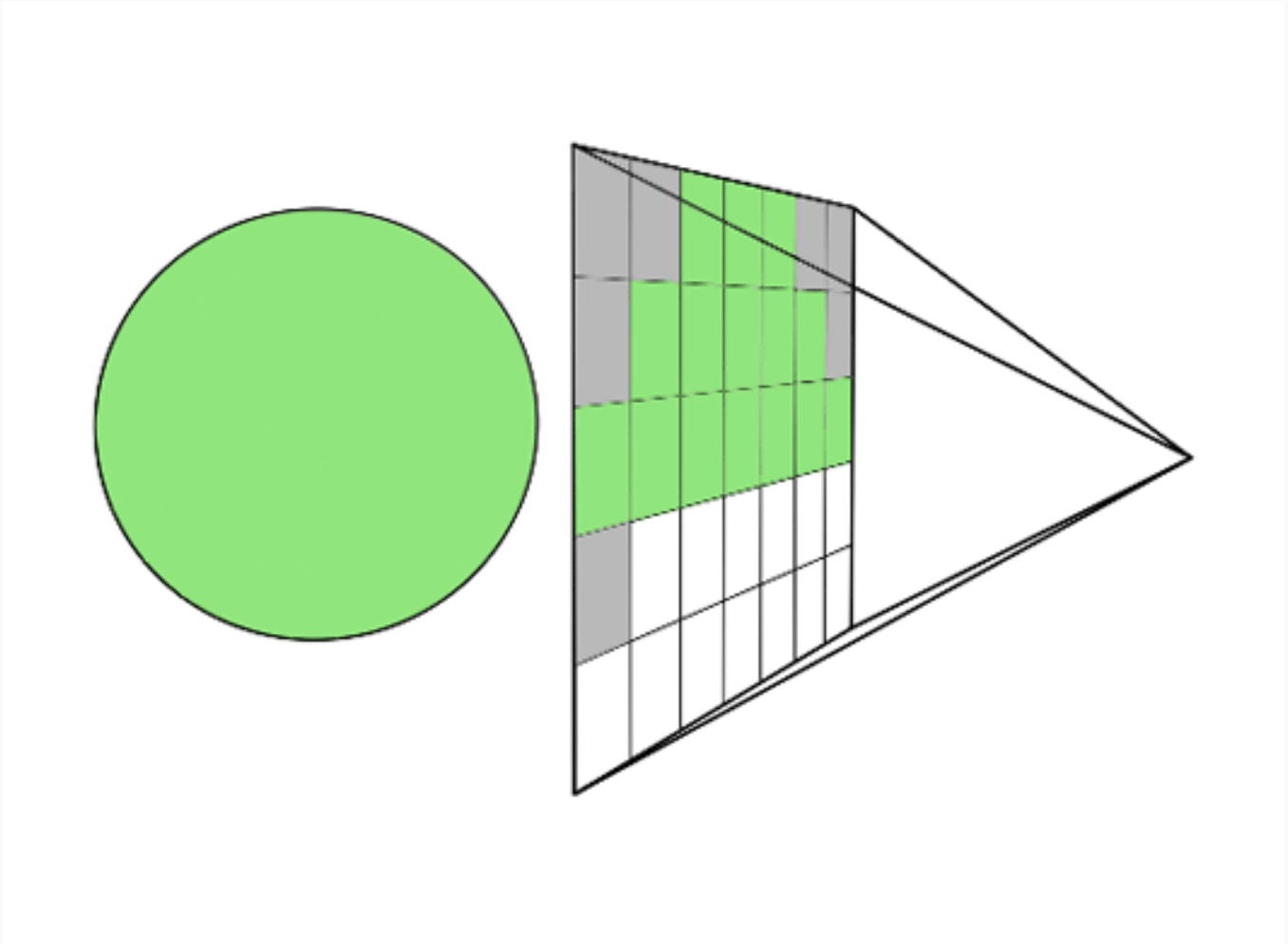
Raycasting Anim



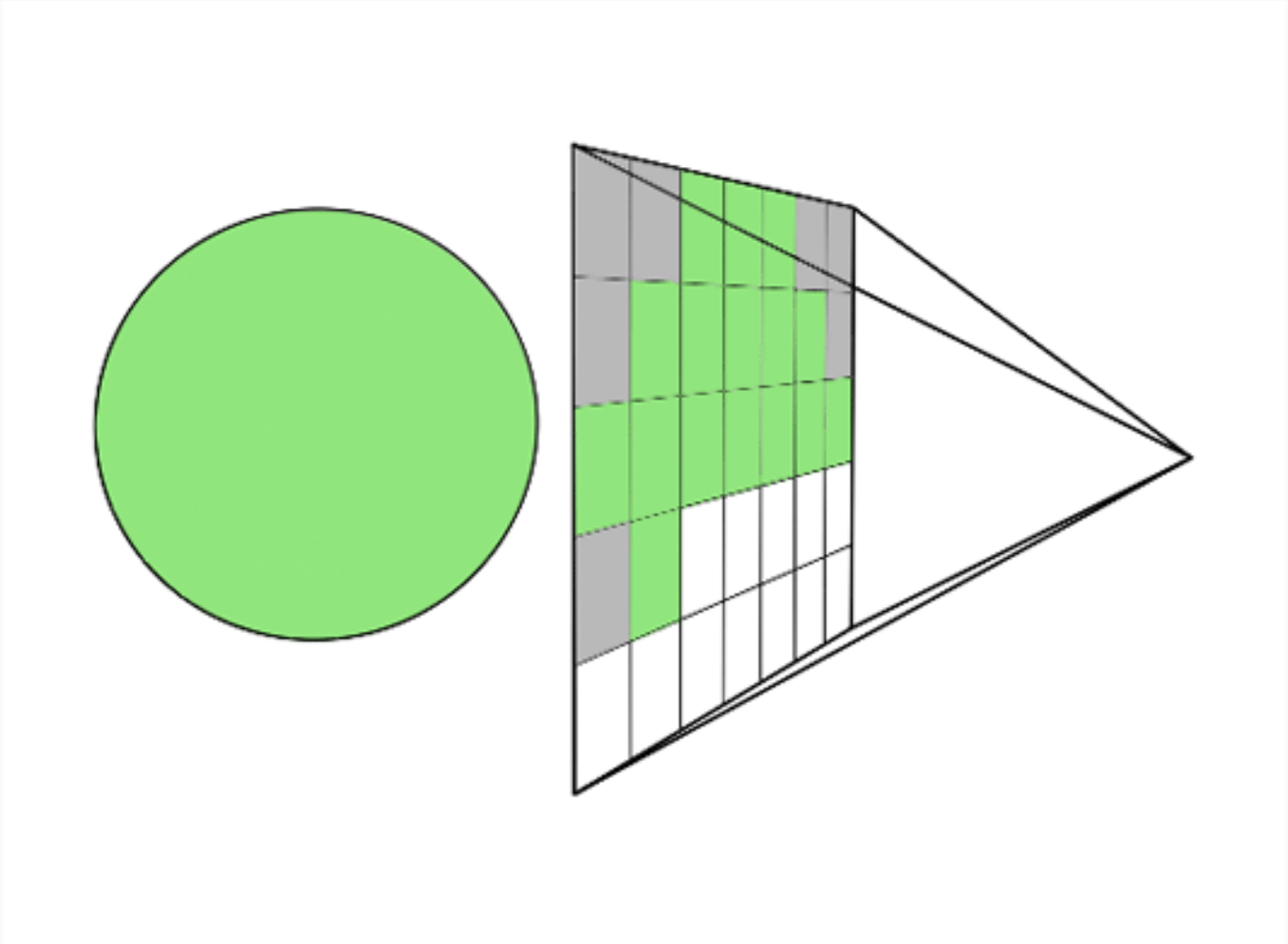
Raycasting Anim



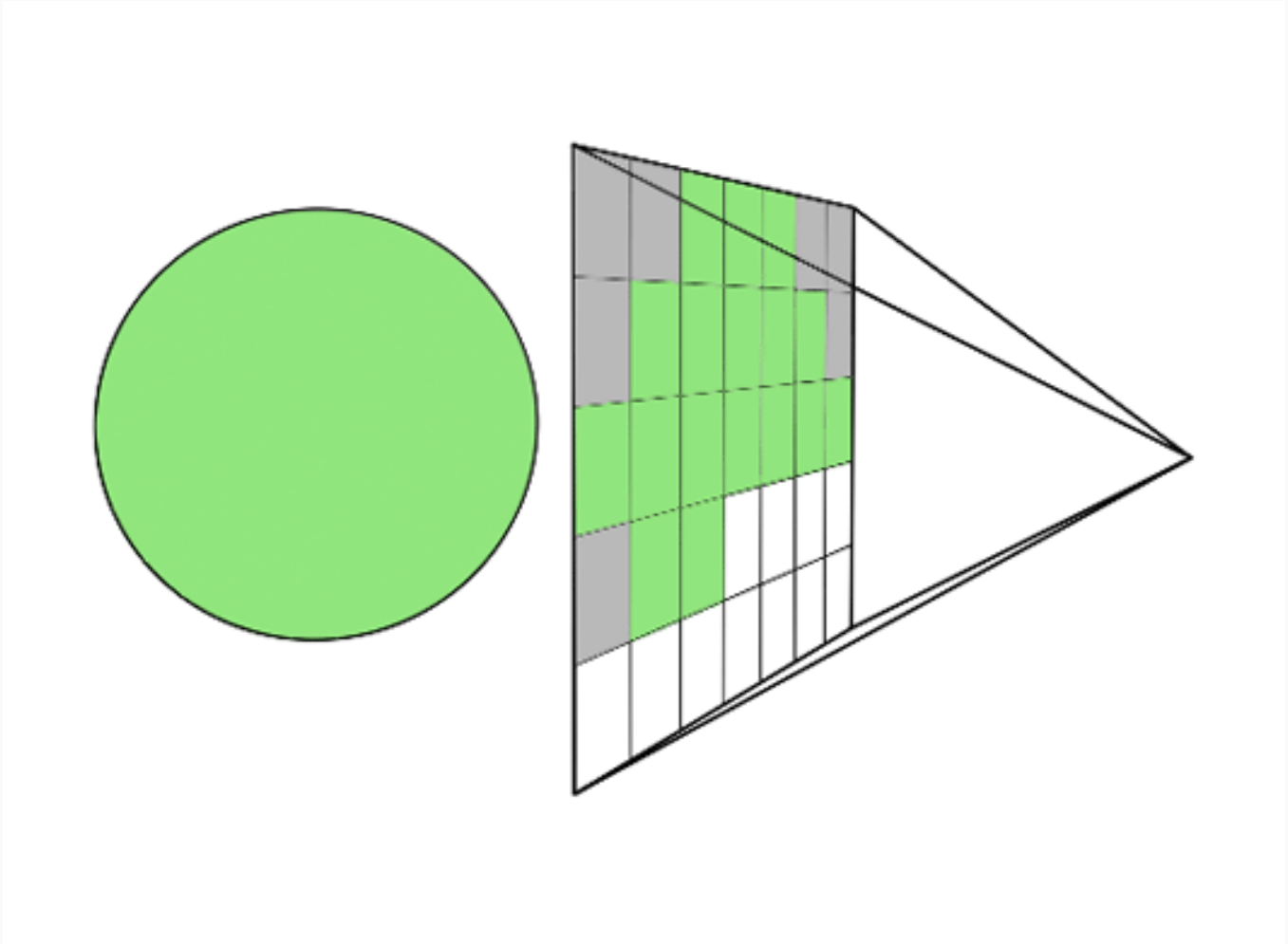
Raycasting Anim



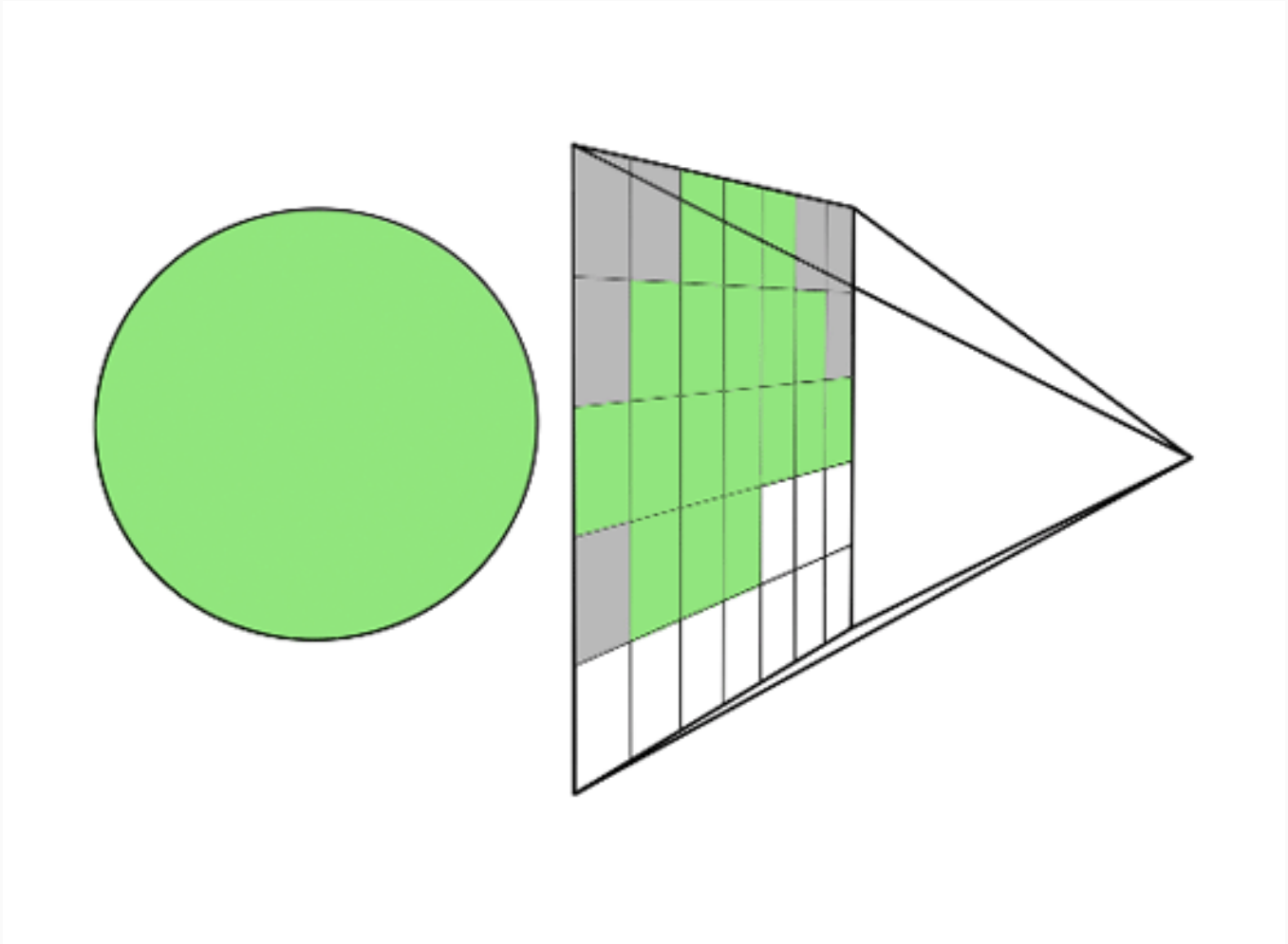
Raycasting Anim



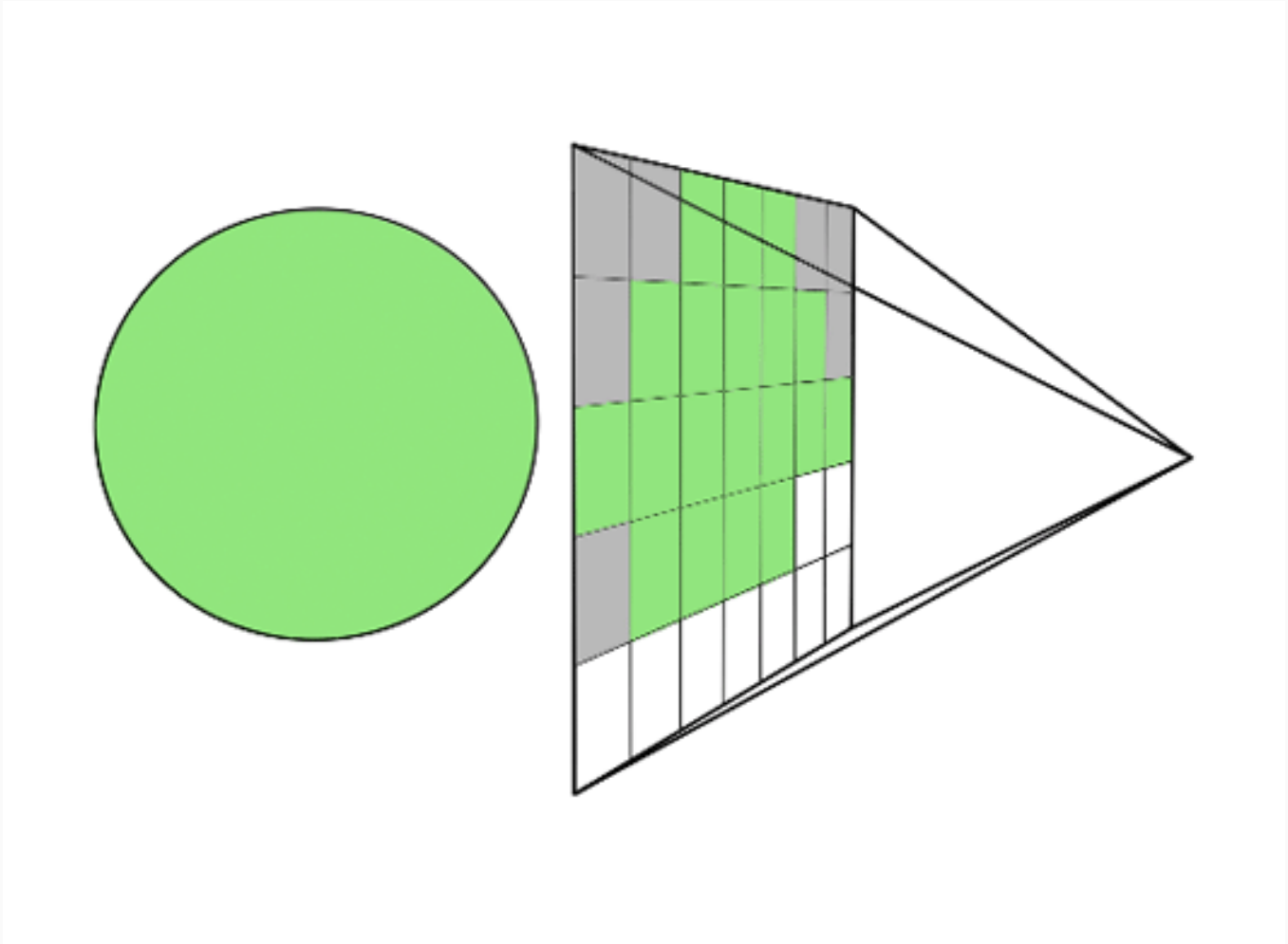
Raycasting Anim



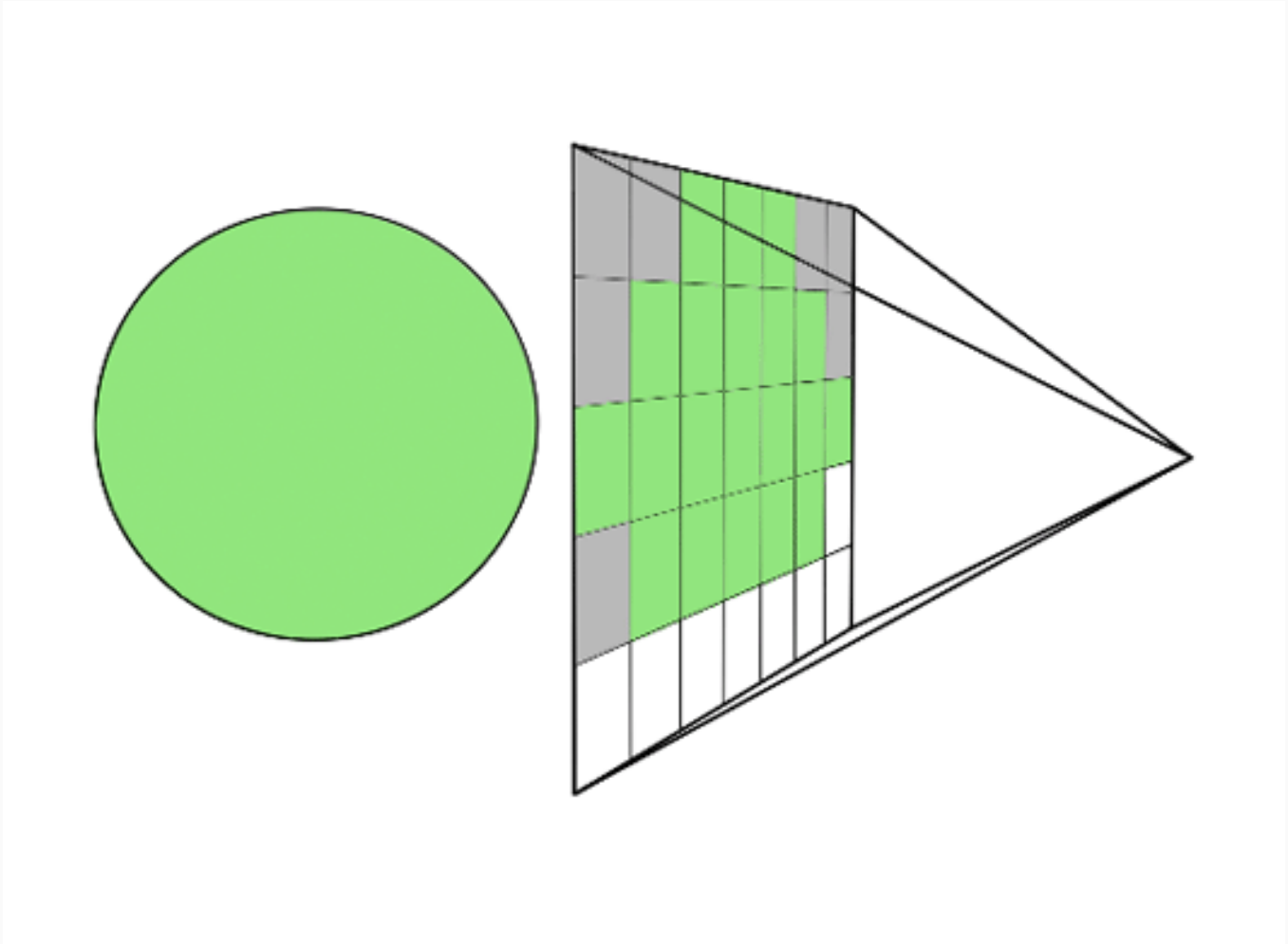
Raycasting Anim



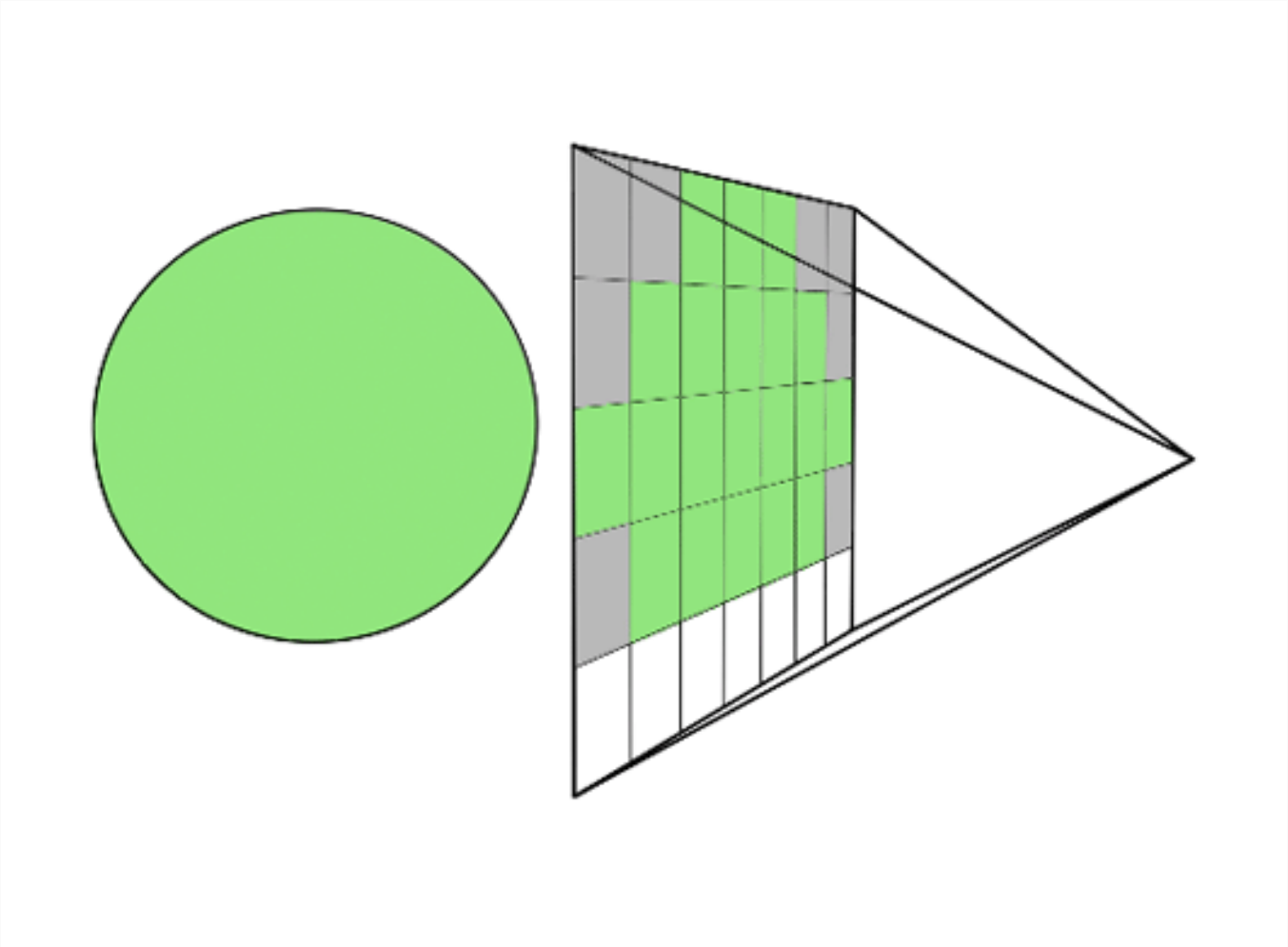
Raycasting Anim



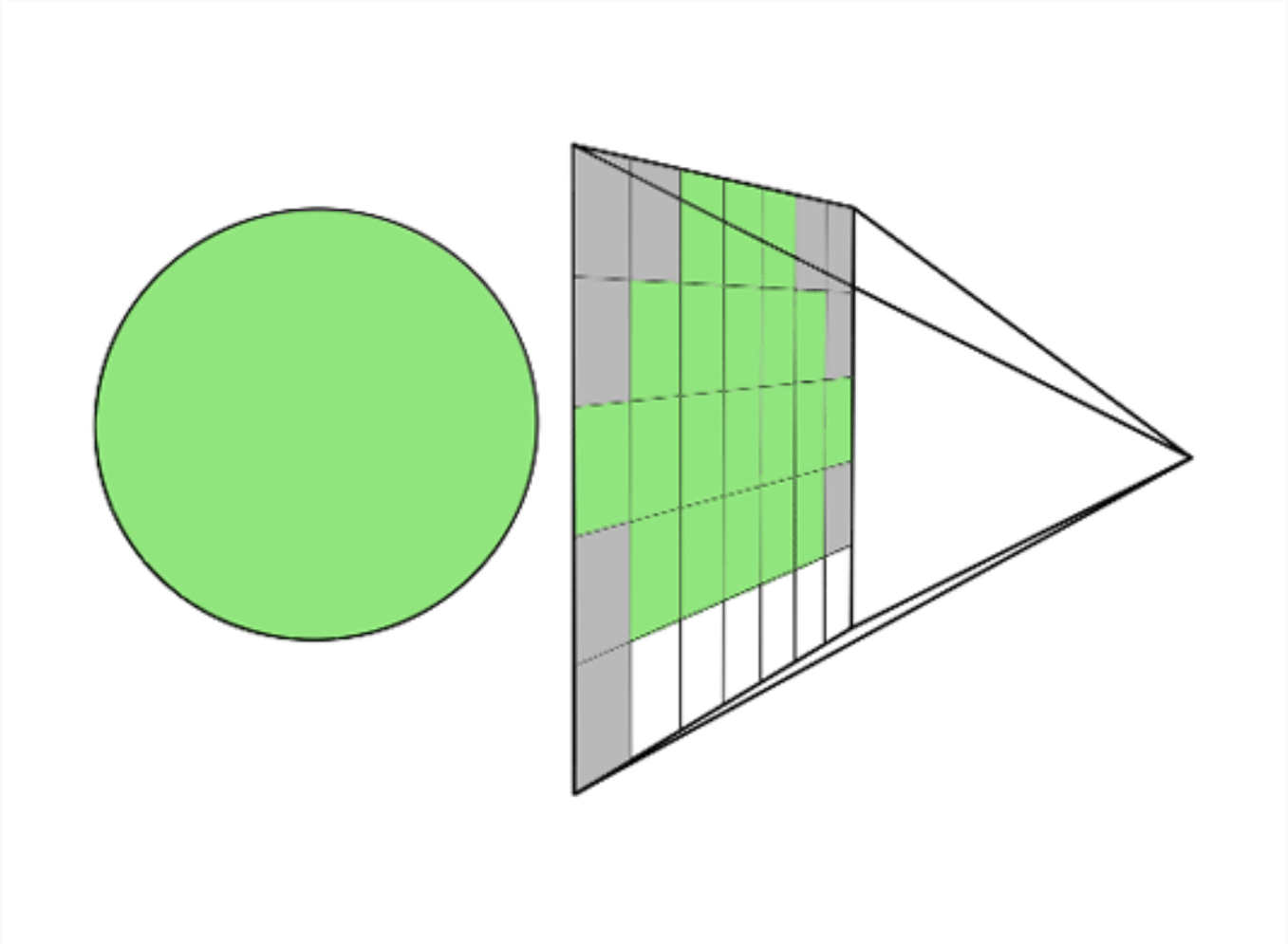
Raycasting Anim



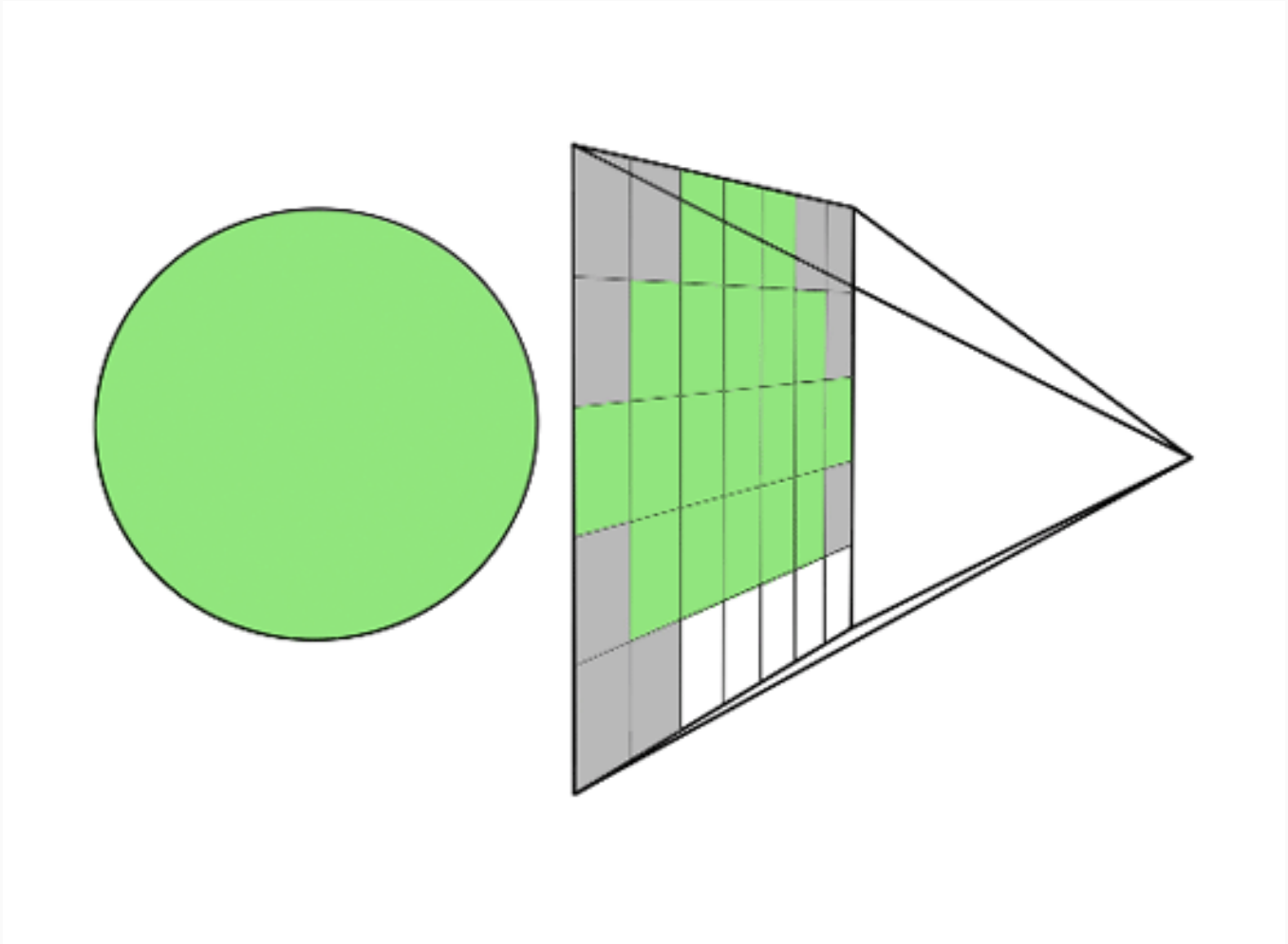
Raycasting Anim



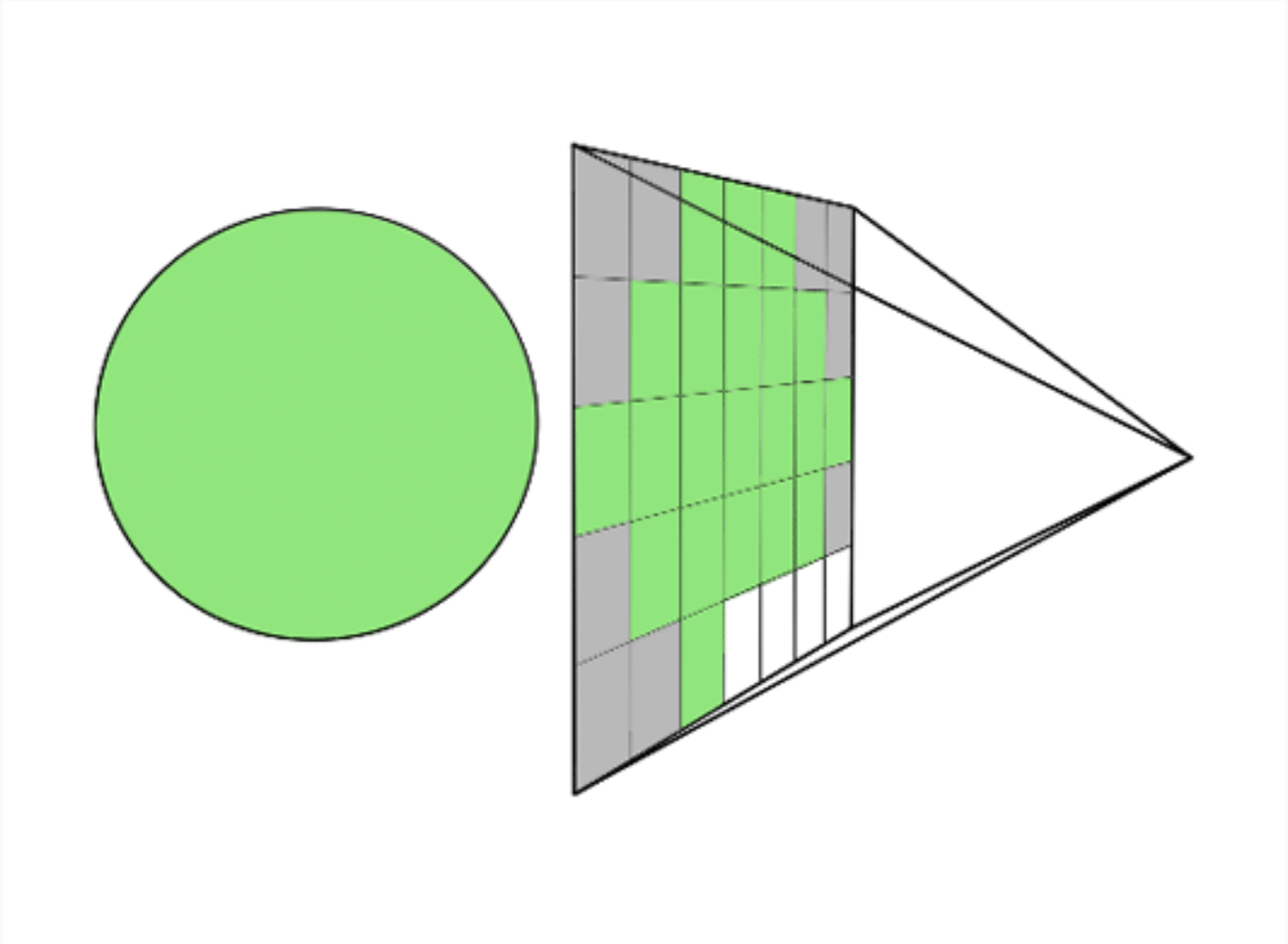
Raycasting Anim



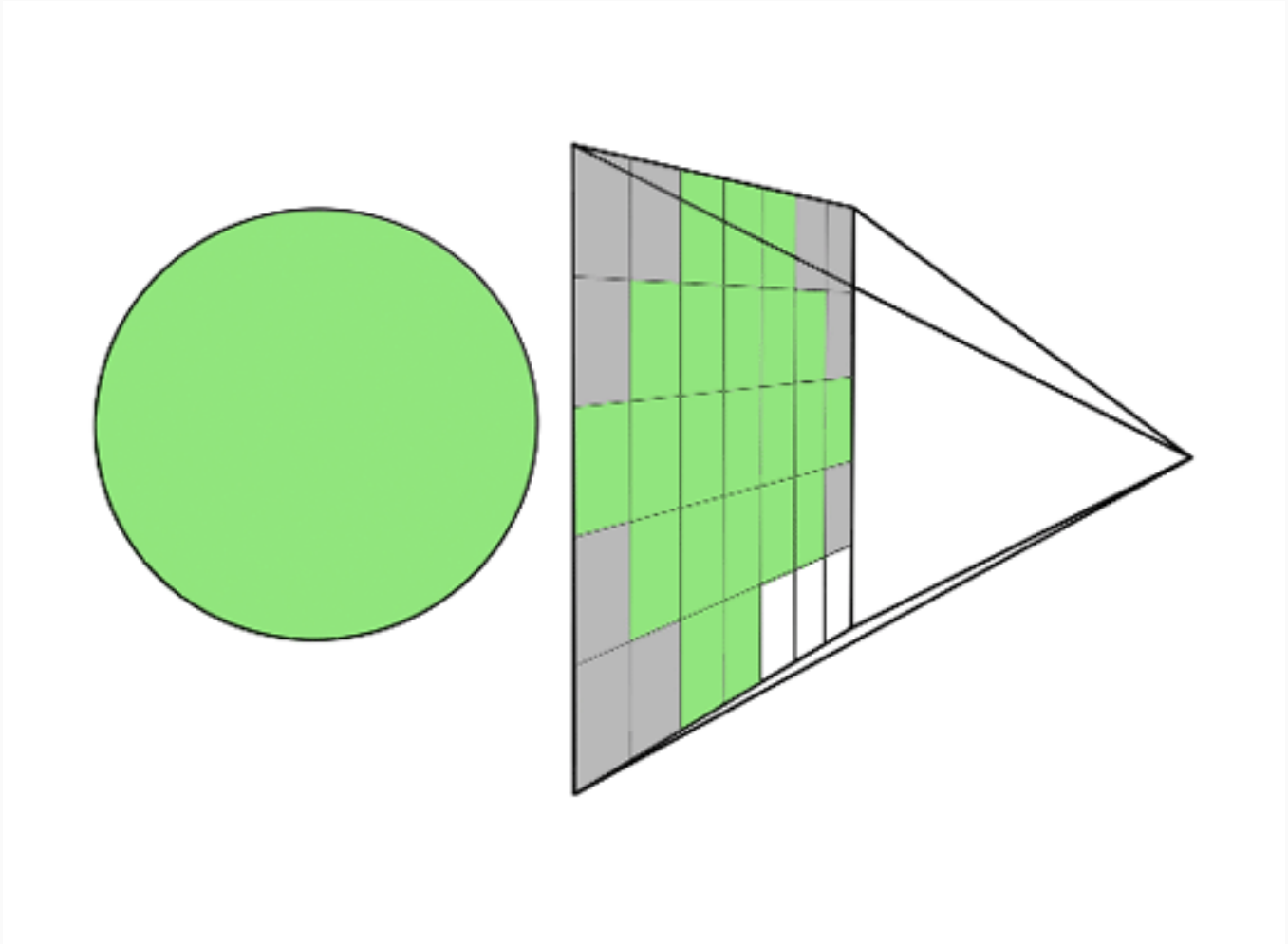
Raycasting Anim



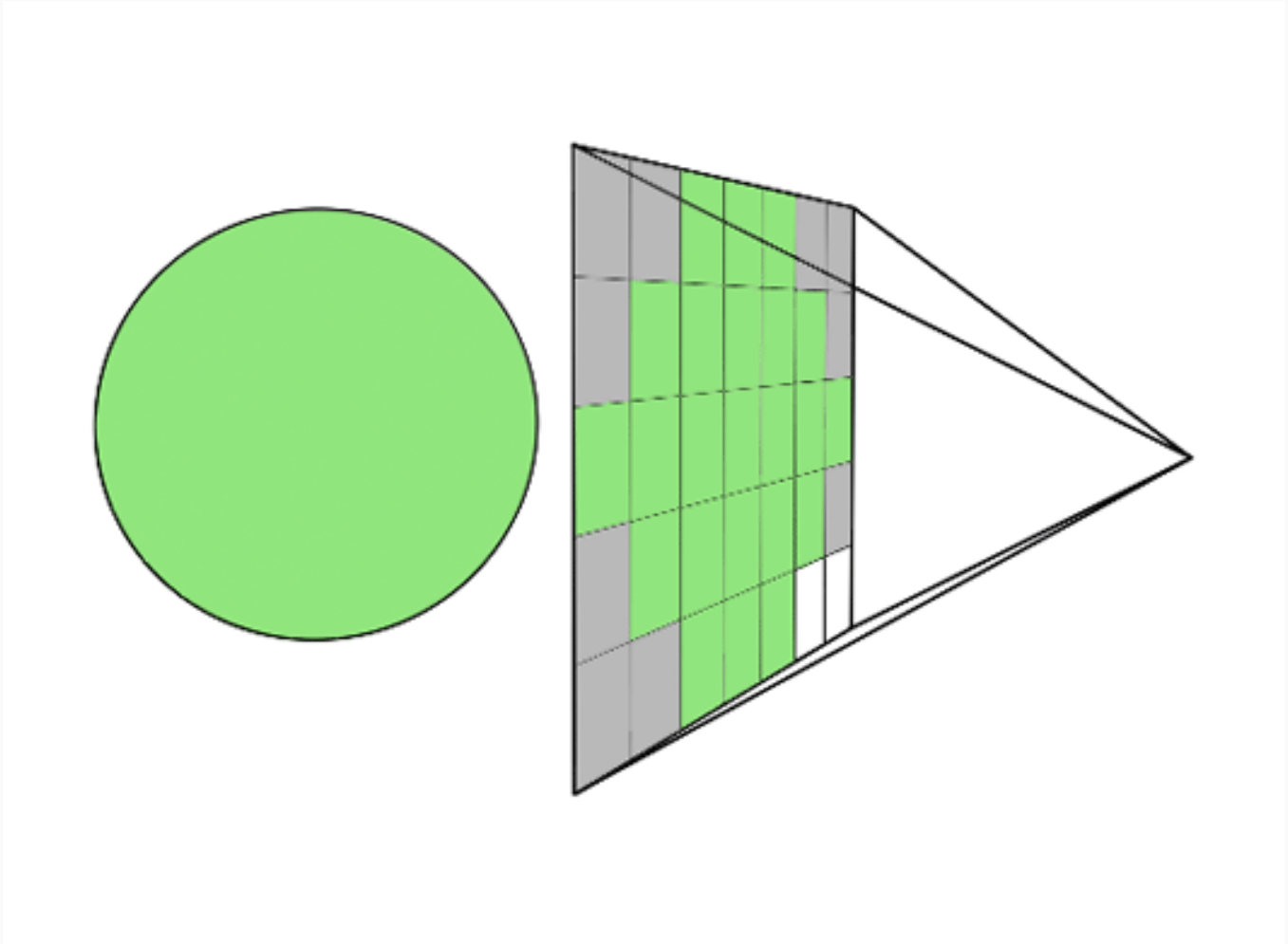
Raycasting Anim



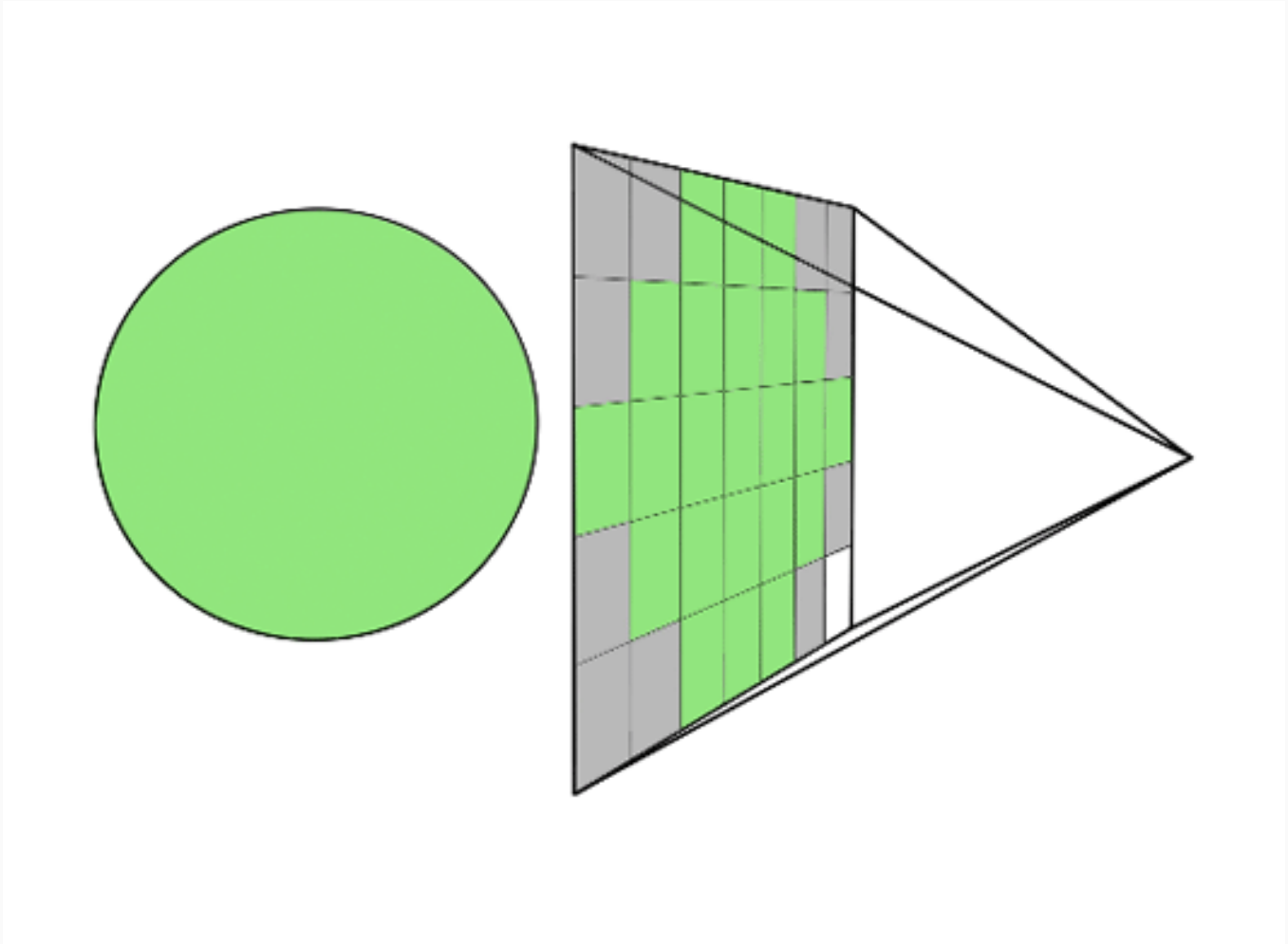
Raycasting Anim



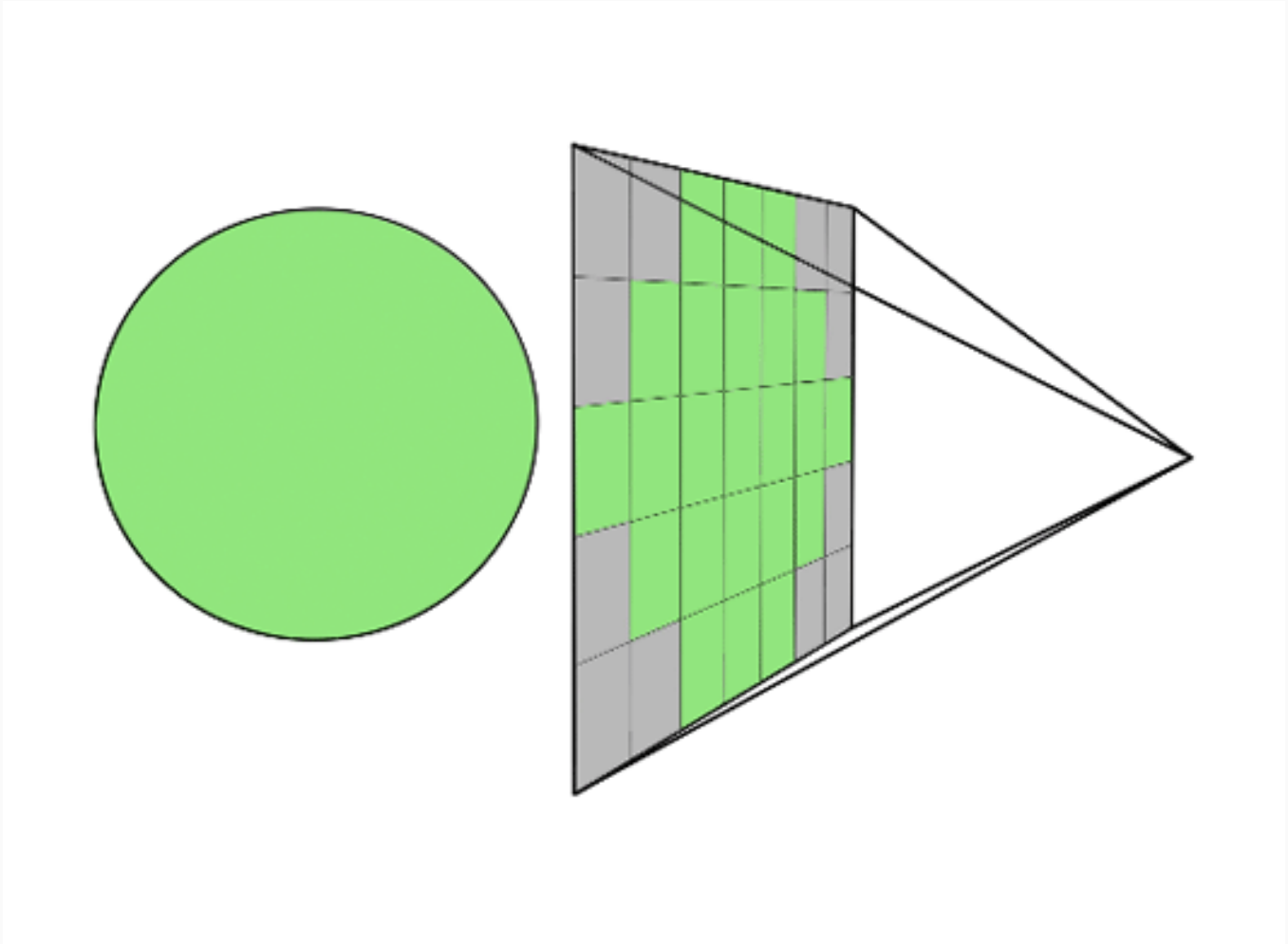
Raycasting Anim



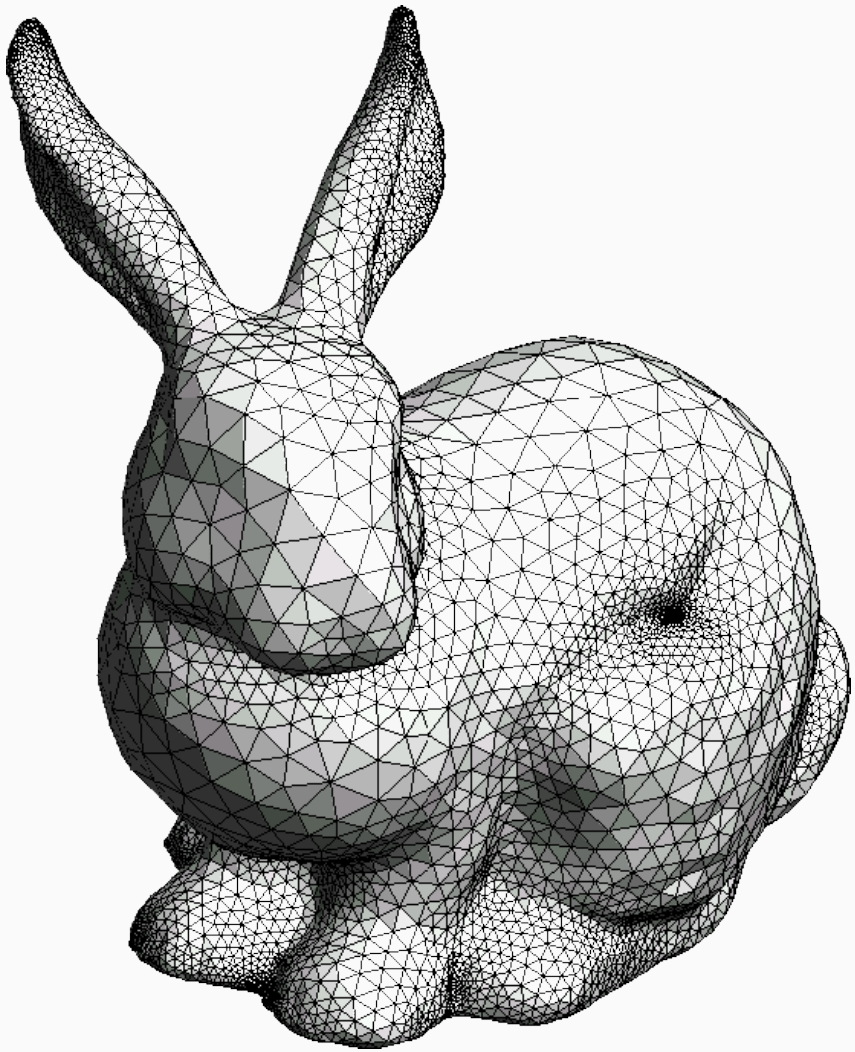
Raycasting Anim



Raycasting Anim



Intersections

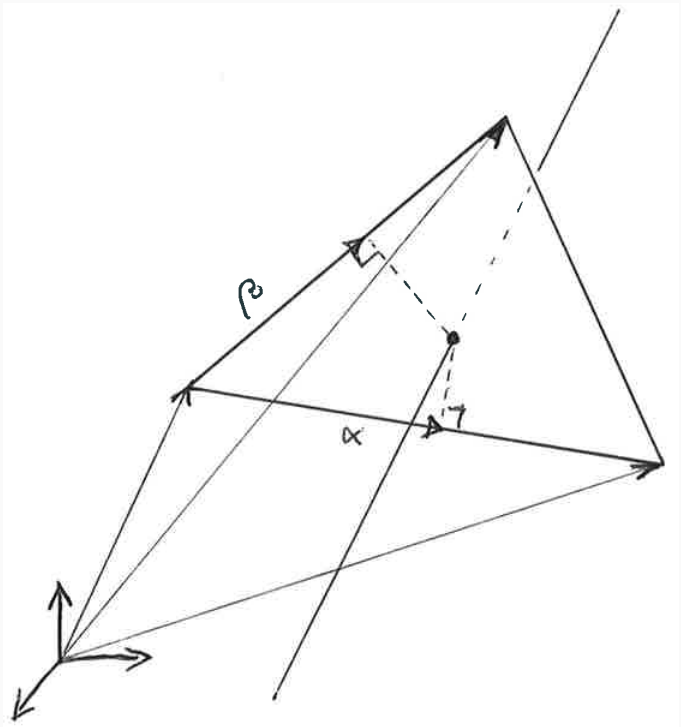


Ray-Triangel Intersection

- Compute two vectors in plane

$$\mathbf{e}_1 = \mathbf{v}_1 - \mathbf{v}_0$$

$$\mathbf{e}_2 = \mathbf{v}_2 - \mathbf{v}_0$$



Ray-Triangel Intersection

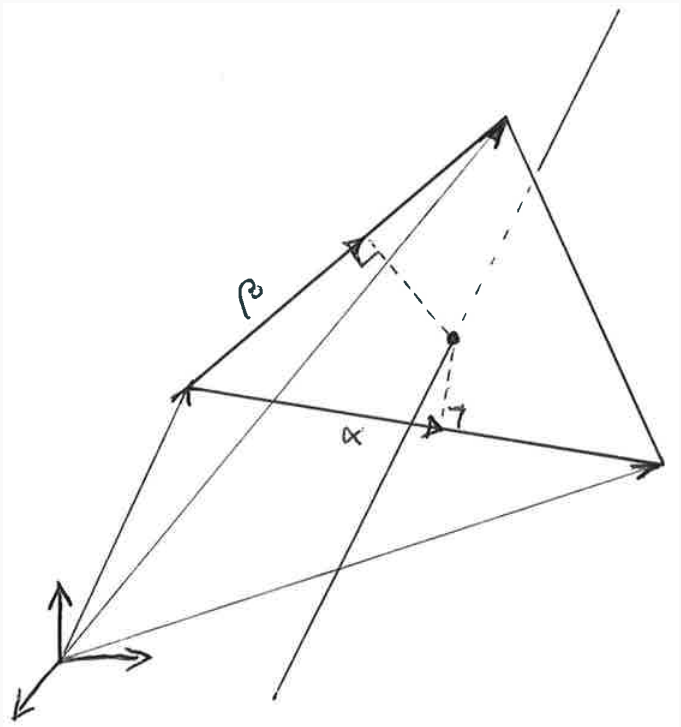
- Compute two vectors in plane

$$\mathbf{e}_1 = \mathbf{v}_1 - \mathbf{v}_0$$

$$\mathbf{e}_2 = \mathbf{v}_2 - \mathbf{v}_0$$

- All points on plane that triangle lies in

$$\mathbf{r} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$



Ray-Triangel Intersection

- Compute two vectors in plane

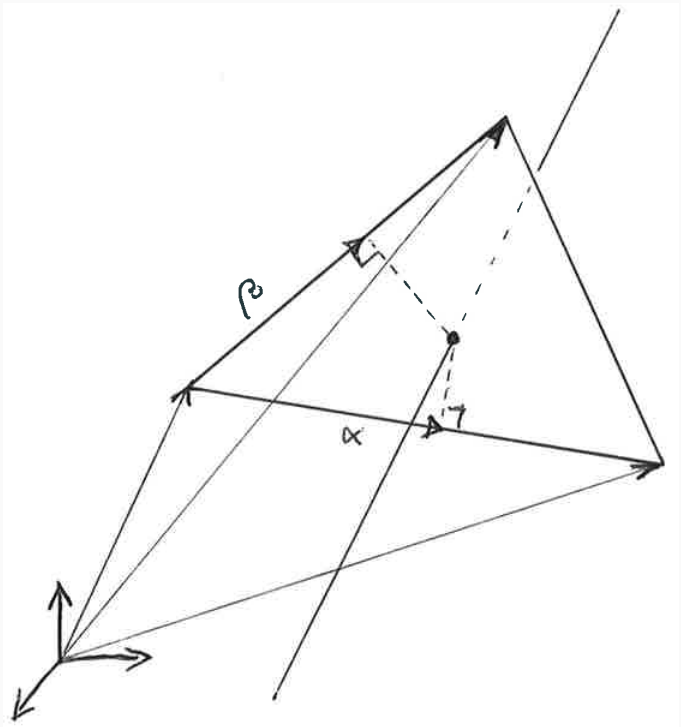
$$\mathbf{e}_1 = \mathbf{v}_1 - \mathbf{v}_0$$

$$\mathbf{e}_2 = \mathbf{v}_2 - \mathbf{v}_0$$

- All points on plane that triangle lies in

$$\mathbf{r} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

- Two degrees-of-freedom



Ray-Triangel Intersection

- Compute two vectors in plane

$$\mathbf{e}_1 = \mathbf{v}_1 - \mathbf{v}_0$$

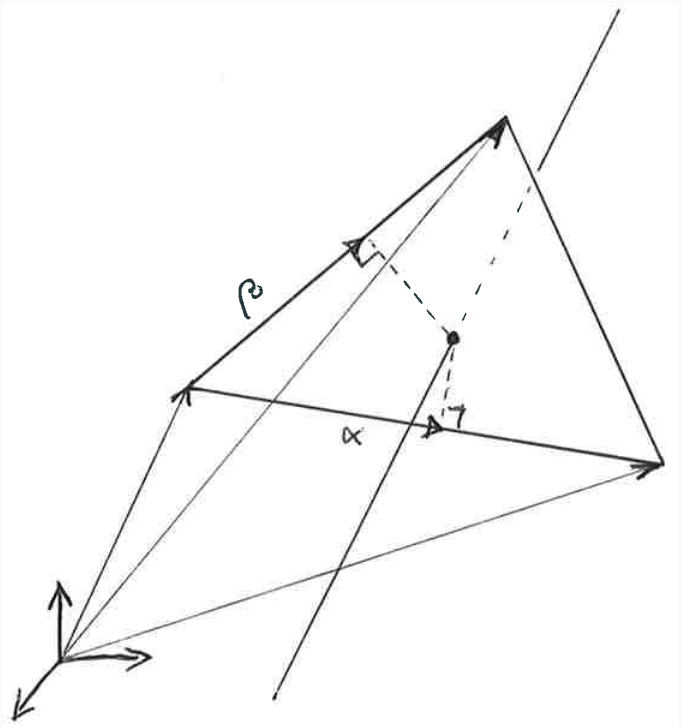
$$\mathbf{e}_2 = \mathbf{v}_2 - \mathbf{v}_0$$

- All points on plane that triangle lies in

$$\mathbf{r} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

- Two degrees-of-freedom
- Inside triangle

$$\{u, v | u \geq 0, v \geq 0, v+u \leq 1\}$$



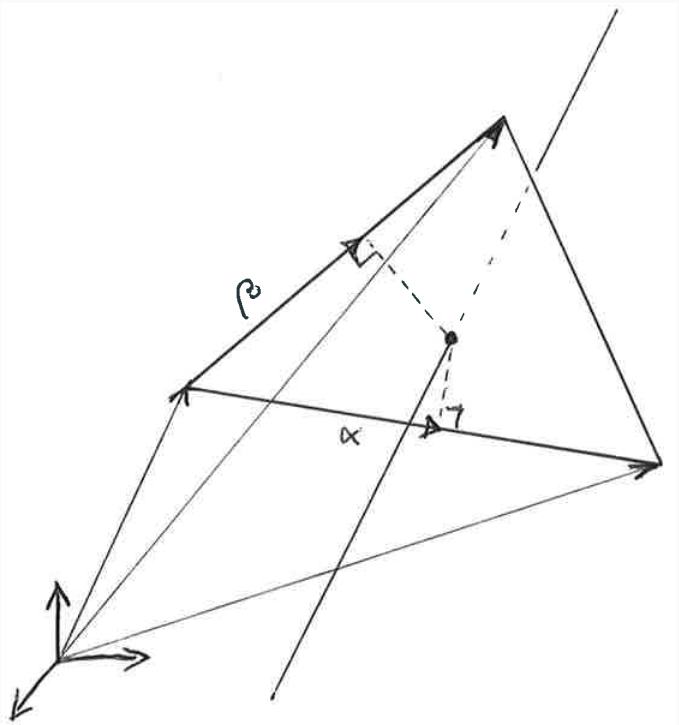
Ray-Triangel Intersection

- Solve for intersection

$$\mathbf{r} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

$$\mathbf{r}_{ij} = \mathbf{s} + t \cdot \mathbf{d}_{ij}$$

$$\mathbf{s} + t \cdot \mathbf{d}_{ij} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$



Ray-Triangel Intersection

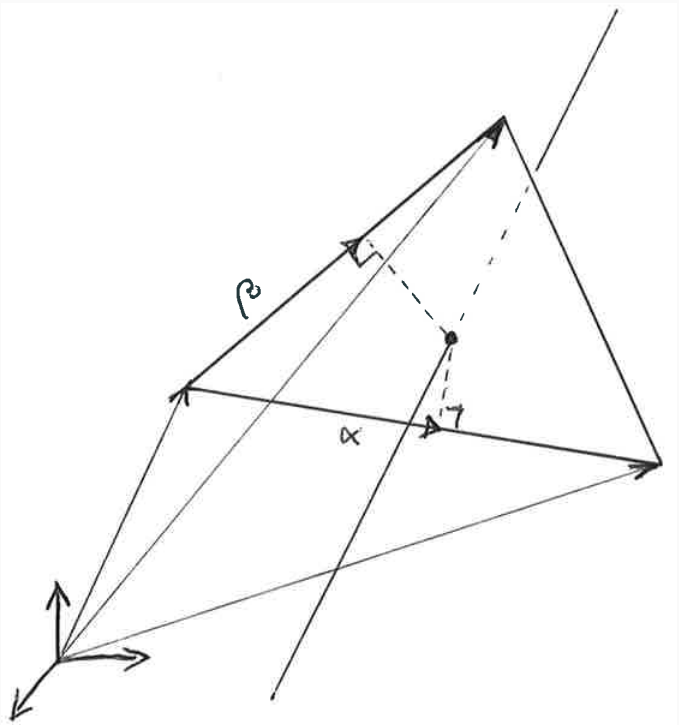
- Solve for intersection

$$\mathbf{r} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

$$\mathbf{r}_{ij} = \mathbf{s} + t \cdot \mathbf{d}_{ij}$$

$$\mathbf{s} + t \cdot \mathbf{d}_{ij} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

- Unknown: $[u, v, t]^{\mathrm{T}} \in \mathbb{R}^3$



Ray-Triangel Intersection

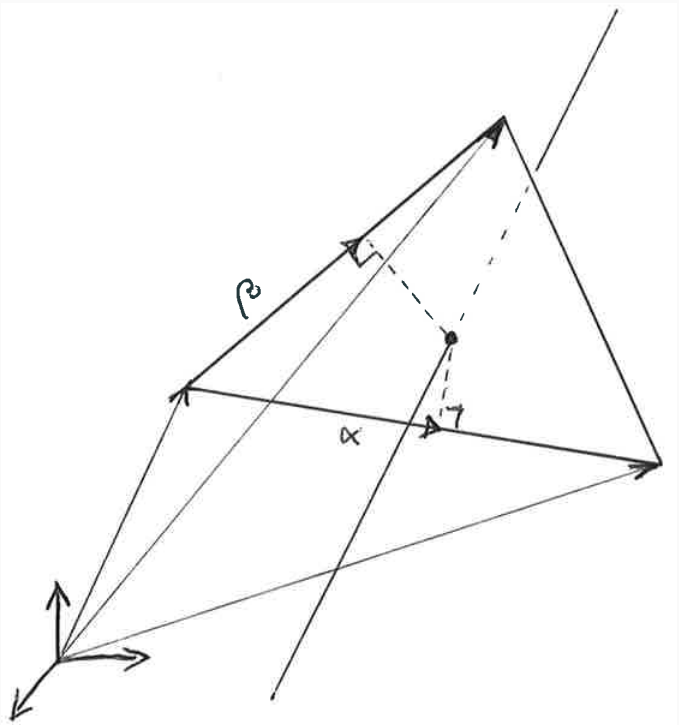
- Solve for intersection

$$\mathbf{r} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

$$\mathbf{r}_{ij} = \mathbf{s} + t \cdot \mathbf{d}_{ij}$$

$$\mathbf{s} + t \cdot \mathbf{d}_{ij} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

- Unknown: $[u, v, t]^{\mathrm{T}} \in \mathbb{R}^3$
- Three equations three unknowns!



Ray-Triangel Intersection

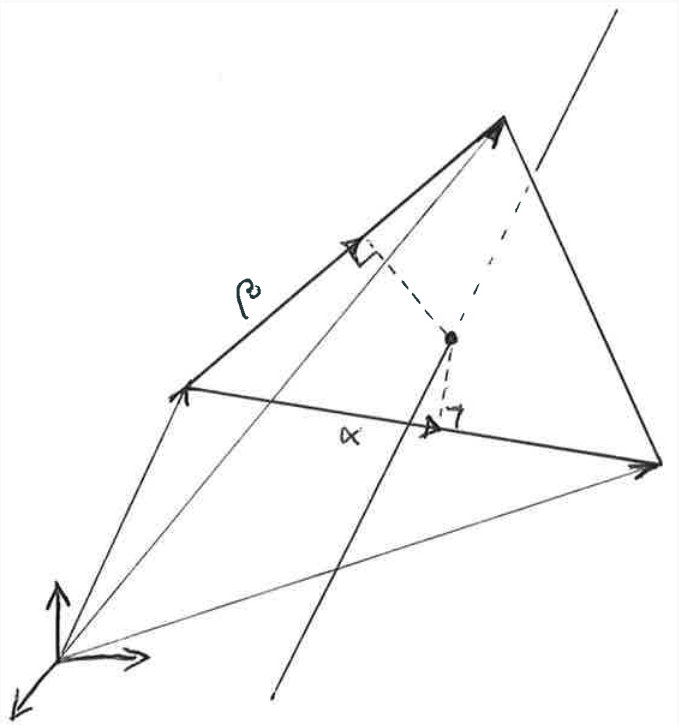
- Solve for intersection

$$\mathbf{r} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

$$\mathbf{r}_{ij} = \mathbf{s} + t \cdot \mathbf{d}_{ij}$$

$$\mathbf{s} + t \cdot \mathbf{d}_{ij} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

- Unknown: $[u, v, t]^{\mathrm{T}} \in \mathbb{R}^3$
- Three equations three unknowns!
- Intersection of plane



Ray-Triangel Intersection

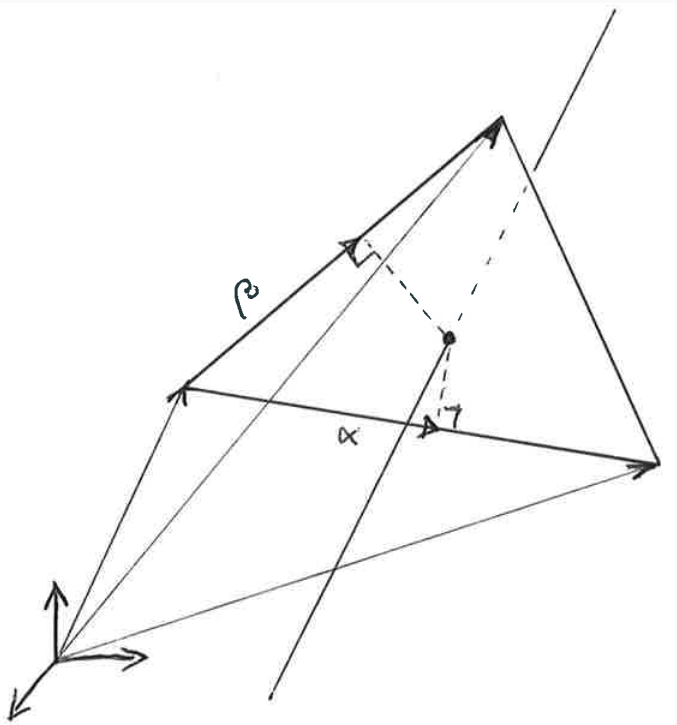
- Solve for intersection

$$\mathbf{r} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

$$\mathbf{r}_{ij} = \mathbf{s} + t \cdot \mathbf{d}_{ij}$$

$$\mathbf{s} + t \cdot \mathbf{d}_{ij} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

- Unknown: $[u, v, t]^{\mathrm{T}} \in \mathbb{R}^3$
- Three equations three unknowns!
- Intersection of plane
- Check boundary conditions of triangle



Ray-Triangle Intersection

- Ray in plane equation

$$\mathbf{s} + t \cdot \mathbf{d}_{ij} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

$$s - \mathbf{v}_0 = -t \cdot \mathbf{d}_{ij} + u\mathbf{e}_0 + v\mathbf{e}_1$$

Ray-Triangle Intersection

- Ray in plane equation

$$\mathbf{s} + t \cdot \mathbf{d}_{ij} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

$$s - \mathbf{v}_0 = -t \cdot \mathbf{d}_{ij} + u\mathbf{e}_0 + v\mathbf{e}_1$$

- Write on matrix form

$$\begin{bmatrix} -d_{ij}^x & e_0^x & e_1^x \\ -d_{ij}^y & e_0^y & e_1^y \\ -d_{ij}^z & e_0^z & e_1^z \end{bmatrix} \cdot \begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} s^x - v_0^x \\ s^y - v_0^y \\ s^z - v_0^z \end{bmatrix}$$

Ray-Triangle Intersection

- Ray in plane equation

$$\mathbf{s} + t \cdot \mathbf{d}_{ij} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

$$s - \mathbf{v}_0 = -t \cdot \mathbf{d}_{ij} + u\mathbf{e}_0 + v\mathbf{e}_1$$

- Write on matrix form

$$\begin{bmatrix} -d_{ij}^x & e_0^x & e_1^x \\ -d_{ij}^y & e_0^y & e_1^y \\ -d_{ij}^z & e_0^z & e_1^z \end{bmatrix} \cdot \begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} s^x - v_0^x \\ s^y - v_0^y \\ s^z - v_0^z \end{bmatrix}$$

- This we know from basic math as

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

Ray-Triangle Intersection

- Ray in plane equation

$$\mathbf{s} + t \cdot \mathbf{d}_{ij} = \mathbf{v}_0 + u\mathbf{e}_0 + v\mathbf{e}_1$$

$$s - \mathbf{v}_0 = -t \cdot \mathbf{d}_{ij} + u\mathbf{e}_0 + v\mathbf{e}_1$$

- Write on matrix form

$$\begin{bmatrix} -d_{ij}^x & e_0^x & e_1^x \\ -d_{ij}^y & e_0^y & e_1^y \\ -d_{ij}^z & e_0^z & e_1^z \end{bmatrix} \cdot \begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} s^x - v_0^x \\ s^y - v_0^y \\ s^z - v_0^z \end{bmatrix}$$

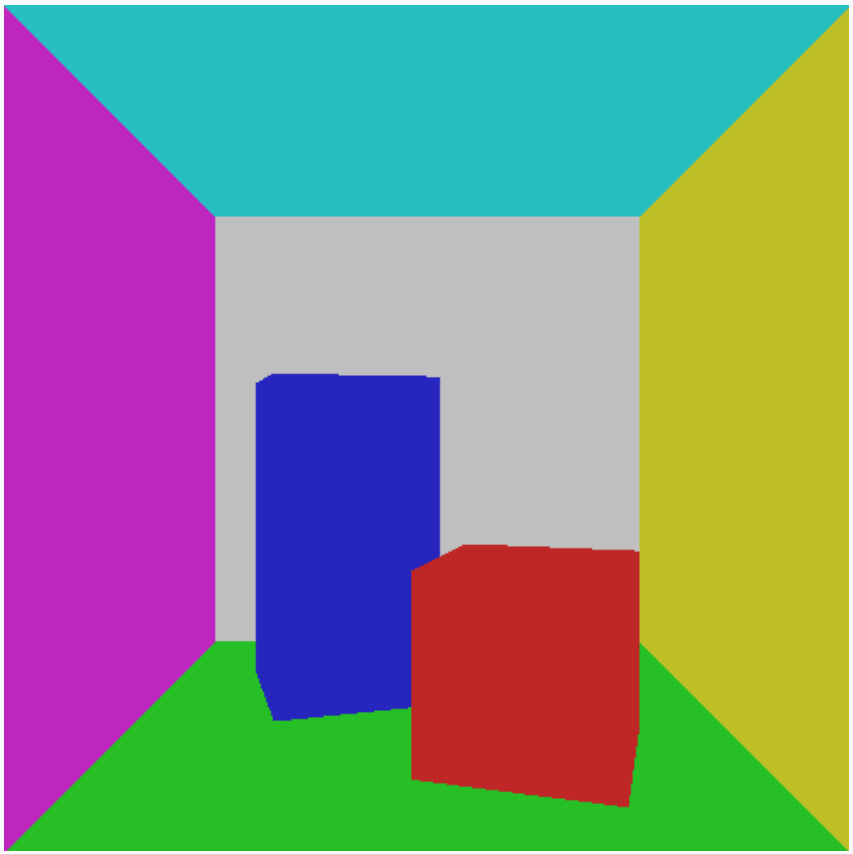
- This we know from basic math as

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

- *silliest solution*

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

Intersections



Ray-Sphere Intersection

- A point $\mathbf{p}$ lies the surface of a sphere with radius r and center $\mathbf{c}$ iff

$$\sqrt{(\mathbf{p}-\mathbf{c})^{\mathrm{T}}(\mathbf{p}-\mathbf{c})}-r=0$$

Ray-Sphere Intersection

- A point $\mathbf{p}$ lies the surface of a sphere with radius r and center $\mathbf{c}$ iff

$$\sqrt{(\mathbf{p}-\mathbf{c})^{\mathrm{T}}(\mathbf{p}-\mathbf{c})}-r=0$$

- Insert ray-equation into sphere

$$\sqrt{(\mathbf{s}+t\mathbf{d}-\mathbf{c})^{\mathrm{T}}(\mathbf{s}+t\mathbf{d}-\mathbf{c})}-r=0$$

$$(\mathbf{s}+t\mathbf{d}-\mathbf{c})^{\mathrm{T}}(\mathbf{s}+t\mathbf{d}-\mathbf{c})=r^2$$

$$t^2+2t(\mathbf{d}^{\mathrm{T}}(\mathbf{s}-\mathbf{c}))+(\mathbf{s}-\mathbf{c})^{\mathrm{T}}(\mathbf{s}-\mathbf{c})-r^2=0$$

Ray-Sphere Intersection

- A point $\mathbf{p}$ lies the surface of a sphere with radius r and center $\mathbf{c}$ iff

$$\sqrt{(\mathbf{p}-\mathbf{c})^{\mathrm{T}}(\mathbf{p}-\mathbf{c})}-r=0$$

- Insert ray-equation into sphere

$$\sqrt{(\mathbf{s}+t\mathbf{d}-\mathbf{c})^{\mathrm{T}}(\mathbf{s}+t\mathbf{d}-\mathbf{c})}-r=0$$

$$(\mathbf{s}+t\mathbf{d}-\mathbf{c})^{\mathrm{T}}(\mathbf{s}+t\mathbf{d}-\mathbf{c})=r^2$$

$$t^2+2t(\mathbf{d}^{\mathrm{T}}(\mathbf{s}-\mathbf{c}))+(\mathbf{s}-\mathbf{c})^{\mathrm{T}}(\mathbf{s}-\mathbf{c})-r^2=0$$

- Quadratic expression i t

$$t=-\mathbf{d}^{\mathrm{T}}(\mathbf{s}-\mathbf{c})\pm\sqrt{(\mathbf{d}^{\mathrm{T}}(\mathbf{s}-\mathbf{c}))^2-(\mathbf{s}-\mathbf{c})^{\mathrm{T}}(\mathbf{s}-\mathbf{c})-r^2}$$

Ray Intersections

- A Raytracer spends most its time doing intersection computations (profile your code)
- Rules of Thumb
  - Can we trivially reject something?
  - Can we re-use computations?
- If we need to do several tests do them in order of computational cost
- Realtime

Ray Intersections

- A Raytracer spends most its time doing intersection computations (profile your code)
- Rules of Thumb
  - Can we trivially reject something?
  - Can we re-use computations?
- If we need to do several tests do them in order of computational cost
- Realtime
- *Its really fun optimisations as the code is often very small and the solutions are really cute*

Ray-Sphere Intersection

- Sphere intersection

$$t = -\mathbf{d}^{\mathrm{T}}(\mathbf{s}-\mathbf{c}) \pm \sqrt{(\mathbf{d}^{\mathrm{T}}(\mathbf{s}-\mathbf{c}))^2 - (\mathbf{s}-\mathbf{c})^{\mathrm{T}}(\mathbf{s}-\mathbf{c}) - r^2}$$

- Look at square-root
  - $= 0$ Only one solution, no need to compute square-root
  - < 0 No real solution, no intersection
  - > 0 We have to compute square root (two intersections)

Ray-Sphere Intersection

- Sphere intersection

$$t = -\mathbf{d}^{\mathrm{T}}(\mathbf{s}-\mathbf{c}) \pm \sqrt{(\mathbf{d}^{\mathrm{T}}(\mathbf{s}-\mathbf{c}))^2 - (\mathbf{s}-\mathbf{c})^{\mathrm{T}}(\mathbf{s}-\mathbf{c}) - r^2}$$

- Look at square-root
  - $= 0$ Only one solution, no need to compute square-root
  - < 0 No real solution, no intersection
  - > 0 We have to compute square root (two intersections)
- *Which of the above cases do we have most of the time?*

Raycaster

Code

```
for (int y=-h/2;y<h/2;y++)
  {
    for(int x=-w/2;x<w/2;x++)
      {
        /*compute primary ray*/
        for(int i=0;i<N_primitives;i++)
          {
            /*1. compute intersection
              2. check closest*/
          }
      }
  }
```

Next Time

Next Time

Lab Start with Lab 1

- Implement a camera structure and infrastructure to deal with transformations

Lecture Thursday 2nd of February

- 11-12 WMB 3.31
- Illumination: Light I
- How to decide colour of pixel

END

eof