Computação Visual

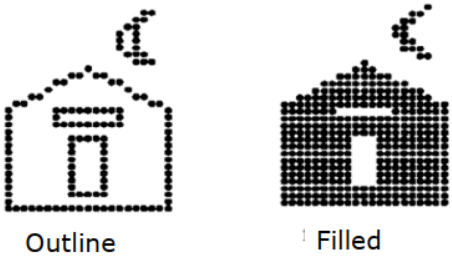
1 - Introdução à Computação Gráfica

- Technology with which picutres, in the broadest sense of the word, are
  - Captured or generated, and presented
  - Manipulated and / or processed
  - Merged with other, non-graphical application data
- Computer graphics means **creation**, **storage** and **manipulation** of **models** and **images**

Wireframe - Draw only lines

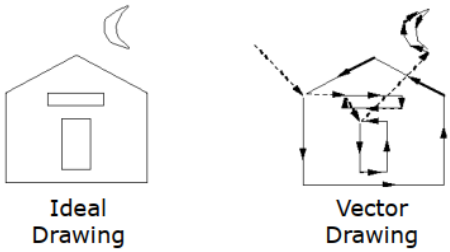
Raster Graphics

- Allows drawing polygons
- Image produced as an array (the **raster**) of picture elements (**pixels**) in the **frame buffer**
- Used in TV displays and laser printers
  - Lowest level of representation. No semantics. Aliasing Errors



Vector Graphics

- Driven by display commands
  - move(x, y); line(x,y)
  - SVG - Scalable Vector Graphics



Interactive Computer Graphics

- User controls **content**, **structure**, and **appearance** of objects and their displayed images, via rapid **visual feedback**
- **Components**
  - Input
  - Processing
  - Output

Batch Computer Graphics

- Non-interactive, **off-line** rendering
- Final production-quality video and film

Data Visualization

- Transformation, selection and / or representation of data with an underlying **geometric structure**

Information Visualization

- Transformation, selection and / or representation of **abstract data**

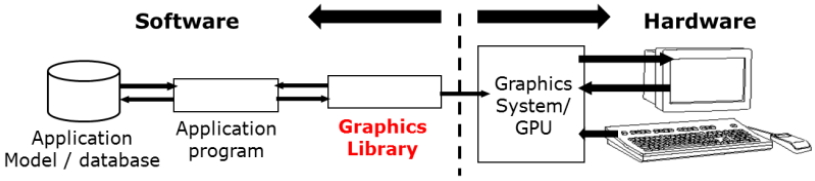
		Output	
		Model	Image
Input	Model	Geometric Modeling	Computer Graphics
	Image	Computer Vision	Image Processing

CG Main Tasks

- **Modeling -** Construct individual models / objects. Assemble them into a 2D or 3D scene
- **Animation -** Static vs. Dynamic scenes. Movement and / or deformation
- **Rendering -** Generate final images. Position of the observer

CG Architecure

- Graphics library / package in **intermediary** between application and display hardware
- Application program **maps / renders** objects / models to images by calling on the **graphics display**
- User **interaction** allows image and / or model modification



Geometric Modeling

- Describes the **shape** of an (real or virtual) object
- Distinguish between **inside**, **outside** and **border** of a model

Modeling

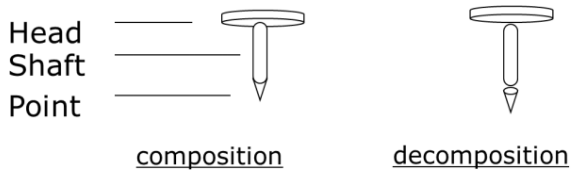
- **Create** models
- Apply **materials** to models
- **Place** models around scene
- Place **lights** in the scene
- Place the **camera**

Rendering

- Take **picture** with the camera

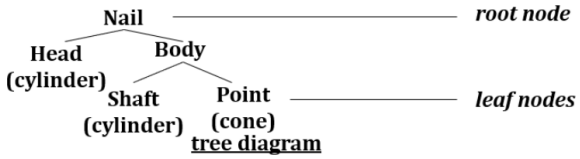
Decomposition of a Geometric Model

- **Hierarchy** of geometrical components
- Reduction to **primitives** (spheres, cubes, etc)



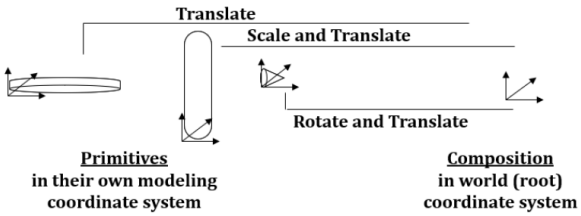
Hierarchical Representation

- Decomposition into collections of primitive shapes
- **Scene Graph -** Data structure to be rendered



Composition of a Geometrical Model

- Assemble primitives to create final object. Apply finite transformations



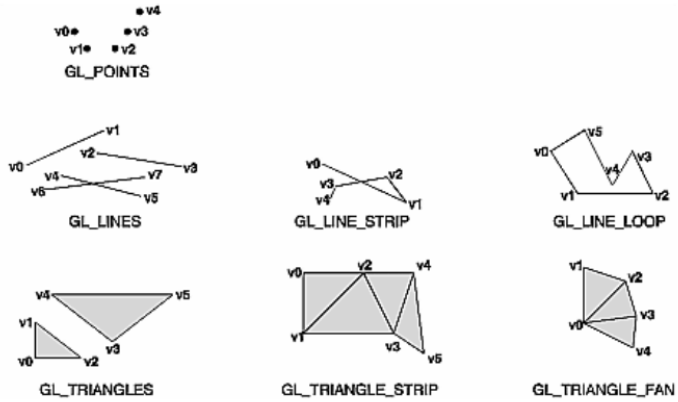
2 - Visualização 2D

GPU - Large collection of highly parallel high-speed arithmetic units

- Run simple programs ("**shaders**")
  - Take in **vertices** and other data
  - Output a **color** value for an individual **pixel**

Geometric Primitives

- **Simple Primitives**
  - Points
  - Line Segments
  - Polygons
- **Geometric Primitives**
  - Parametric curves / surfaces
  - Cubes, spheres, cylinders



Attributes

- Determine the appearance of primitives
  - Color
  - Size and width
  - Stipple pattern
  - Polygon mode - Display as **filled**, **edges** or **vertices**

General Polygons

- Define as **sets of triangles**

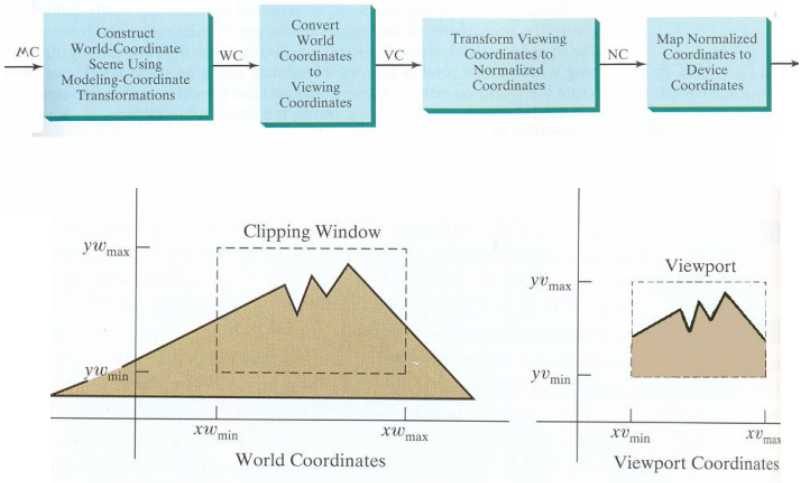
Triangles

- **Simple -** Edges cannot cross
- **Convex -** All points on a line segment between two points on a triangle also belong to the triangle
- **Flat -** All triangle points belong to the same plane

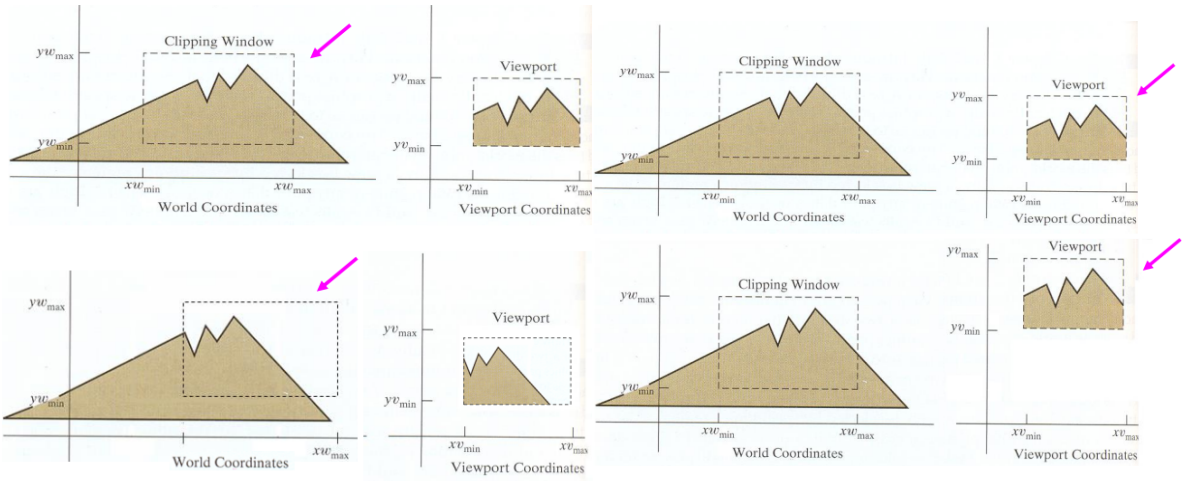
Application program must **tesselate** a polygon into triangles

2D Visualization

- Define a 2D scene in the **world coordinate system**
- Select a **clipping window** in the XOY plane
  - The window contents will be displayed
- Select a **viewport** in the display
  - The viewport displays the contents of the clipping window

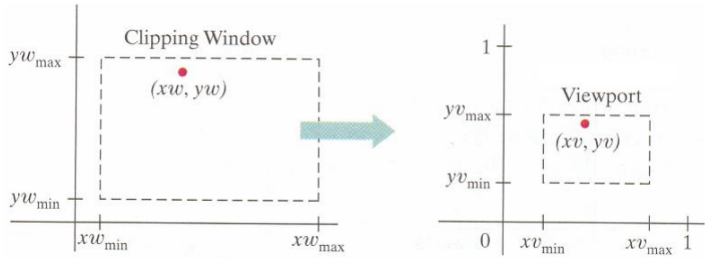


Moving the Clipping Window Moving the Viewport



Window to Viewport Transformation

- Given a **clipping window** and a **viewport**



- Point **(xw, yw)** is transformed into point **(xv, yv)**

Keep Relative Lengths Scaling Factors and Displacements

Taking into account the distance ratios

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

Solving for the unknowns

$$xv = s_x\, xw + t_x$$

$$yv = s_y\, yw + t_y$$

The scaling factors are

$$sx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} \qquad sy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

Global displacements are given by

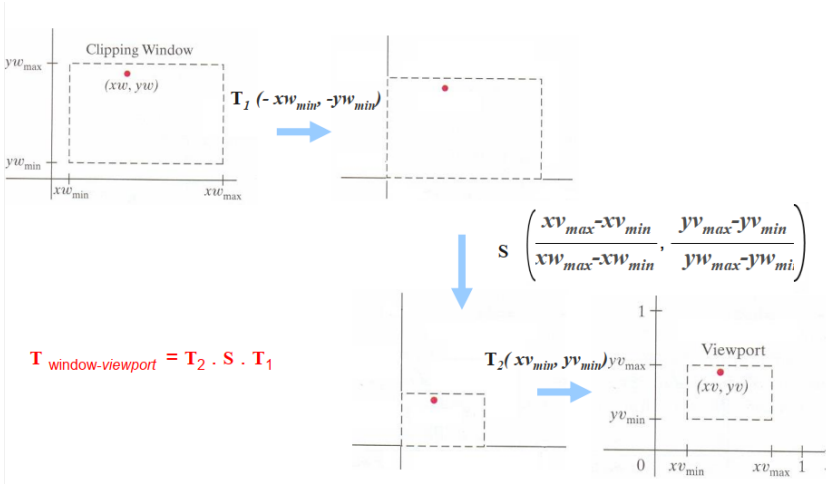
$$t_x = \frac{xw_{max}xv_{min} - xw_{min}xv_{max}}{xw_{max} - xw_{min}}$$

$$t_y = \frac{yw_{max}yv_{min} - yw_{min}yv_{max}}{yw_{max} - yw_{min}}$$

- As an alternative, the window-to-viewport transformation can be defined as a **sequence of transformations which converts a** rectangular **clipping window** into a rectangular **viewport**
  - **1** - **Scaling** of the clipping window to the size of the viewport with **fixed point** (xwmin, ywmin)
  - **2** - **Translation** of **(xwmin, ywmin)** to **(xvmin, yvmin)**

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & xw_{\min}(1-s_x) \\ 0 & s_y & yw_{\min}(1-s_y) \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix} 1 & 0 & xv_{\min} - xw_{\min} \\ 0 & 1 & yv_{\min} - yw_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{\text{window, normviewp}} = \mathbf{T} \cdot \mathbf{S} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$



- **Default** definitions are mostly used!
- Square clipping window. (-1, -1) and (+1, +1)
- Viewport occupies the whole display window
- Changes can be made
  - The viewport might occupy only part of the display window
  - The clipping window can be larger or smaller
- Pay attention to the **aspect ratios**

3 - Transformações 2D

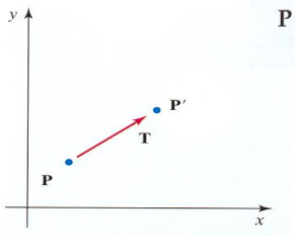
- **Position**, **orientation** and **scaling** for objects in XOY
- Representation using **matrices**

Translation

- We need the displacement values in **x** and **y**

$$x' = x + t_x, \qquad y' = y + t_y$$

$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}, \qquad \mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$



$$\mathbf{P}' = \mathbf{P} + \mathbf{T}$$

- Each object is displaced without any deformation. **Rigid body** transformation
- To displace a **line segment**, apply the transformation to the **two end points**
- To displace a **polygon**, apply the transformation to the polygon's **vertices**

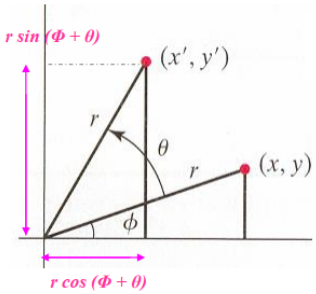
Rotation

- We need the **center of rotation** and a **rotation angle** (positive -» counter-clockwise)

Rotation around the origin

$$x' = r\cos(\Phi+\theta) = r\cos\Phi\cos\theta - r\sin\Phi\sin\theta$$

$$y' = r\sin(\Phi+\theta) = r\cos\Phi\sin\theta + r\sin\Phi\cos\theta$$



Original point coordinates in polar coordinates:

$$x = r\cos\Phi$$

$$y = r\sin\Phi$$

Replacing in the above equations, we get the desired result:

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

- If a point is represented by a row vector, the multiplication order is changed and the corresponding rotation matrix is the transpose

$$P' = R \cdot P \quad \text{com} \quad R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Rotation around an arbitrary point

- Consider this transformation as being made up of a **sequence of elementary transformations**

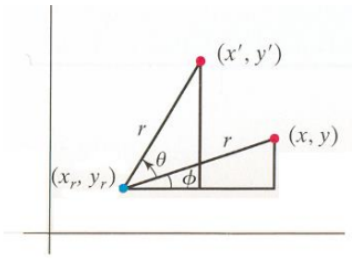
$$x' = r\cos(\theta+\Phi) + x_r \qquad x = r\cos\Phi + x_r$$
$$y' = r\,sen(\theta+\Phi) + y_r \qquad y = r\,sen\,\Phi + y_r$$

$$x' = r\cos\theta\ \cos\Phi - r\ sen\ \theta\ sen\ \Phi + x_r$$

$$y' = r\cos\theta\ sen\ \Phi + r\,sen\,\theta\ \cos\Phi + y_r$$

$$x' = (x - x_r)\cos\theta - (y - y_r)\,sen\,\theta + x_r$$

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$



Scaling

- Applied to change the size of an object with **Sx** and **Sy** as **scaling factors**
- Scaling factors are positive **S > 0**

$$x' = x \cdot s_x$$

$$y' = y \cdot s_y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

transformation matrix

$$P' = S \cdot P$$

Obtaining a larger square through a scaling transformation, s_x=2, s_y=2

- **Sx = Sy** -» **uniform** scaling
- **Sx \= Sy** -» **non-uniform** scaling

The scaled objects are **repositioned** if **not** originally **centered** of the coordinates' origin

- **S < 1** -» it will be **closer** to the origin
- **S > 1** -» it will be **farther** from the origin

Scaling relative to a fixed point

- We can control the position of the object by choosing a **fixed point** that remains **unchanged**

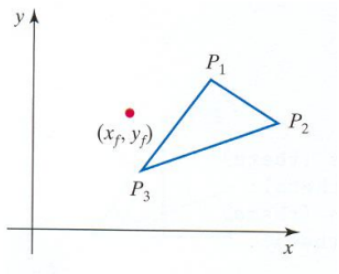
$$x' - x_f = (x - x_f) \cdot s_x$$

$$y' - y_f = (y - y_f) \cdot s_y$$

$$x' = x \cdot s_x + x_f(1 - s_x)$$

$$y' = y \cdot s_y + y_f(1 - s_y)$$

constant for every point



Homogeneous Coordinates

- To carry out **sequences of transformations**, each transformation is represented as a **matrix** using **homogeneous coordinates**

The three **basic transformations** can be represented as

$$\mathbf{P' = M_1 \cdot P + M_2}$$

M1 is a **2x2 matrix**, **M2** is a **column vector**

A more efficient representation uses **just one matrix** which

- Can **represent all the transformations** in a sequence
- Is **applied just once** to every point

A single **3x3 matrix** represents all multiplicative and additive terms. The 3rd column represents the displacement (additive) factors

Every point is now represented by 3 coordinates

$$(x, y) \rightarrow (x_h, y_h, h), \quad h \neq 0$$

$$x = x_h / h \qquad y = y_h / h$$

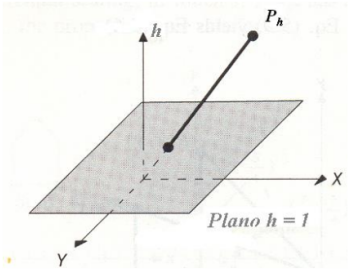
$$(x.h, y.h, h)$$

- An easy choice is:

  $h = 1$

- Which implies:

$$(x, y) \rightarrow (x, y, 1)$$



- Indefinite number of points **Ph** in the 3D homogeneous space that correspond to a single point **(x,y)**

2D Translation 2D Rotation 2D Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P} \qquad \mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P} \qquad \mathbf{P}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{P}$$

Inverse Transformations

- **Translation -** The inverse of a translation is a translation with **symmetrical parameters** in **x** and **y**

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

- **Rotation -**The inverse rotation is obtained by using the **symmetrical rotation angle**. The inverse rotation matrix is the **transpose** of the original rotation matrix. Only the **sinus** are affected

$$\mathbf{R}^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (\mathbf{R}^{-1} = \mathbf{R}^T)$$

- **Scaling -** For the **inverse scaling matrix**, replace the each scaling factor **S** by **1/S**

$$\mathbf{S}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The **product** of any **matrix M**, representing a given transformation, by its **inverse transformation** results in the **identity matrix**

$$\mathbf{M} \cdot \mathbf{M}^{-1} = \mathbf{I}$$

Concatenation of Transformations

- We can compute the matrix representing a sequence of transformations by **multiplying the matrices representing the individual transformations**, in the appropriate order

second transformation to be applied — first transformation to be applied

$$\mathbf{P'} = \mathbf{M}_2 \cdot \mathbf{M}_1 \cdot \mathbf{P}$$
$$= \mathbf{M} \cdot \mathbf{P}$$

Concatenation of 2 Translations Concatenation of 2 Scalings

$$\mathbf{P}' = \mathbf{T}(t_{2x}, t_{2y}) \cdot \{\mathbf{T}(t_{1x}, t_{1y}) \cdot \mathbf{P}\}$$
$$= \{\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y})\} \cdot \mathbf{P}$$

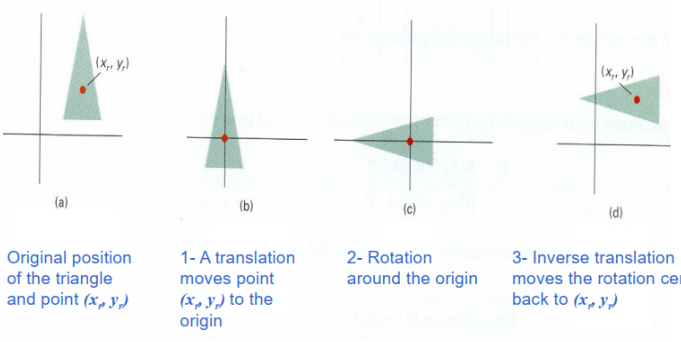
$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y}) = \mathbf{T}(t_{1x} + t_{2x}, t_{1y} + t_{2y})$$

$$\begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x} \cdot s_{2x} & 0 & 0 \\ 0 & s_{1y} \cdot s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{S}(s_{2x}, s_{2y}) \cdot \mathbf{S}(s_{1x}, s_{1y}) = \mathbf{S}(s_{1x} \cdot s_{2x}, \; s_{1y} \cdot s_{2y})$$

Rotation around an arbitrary point



Original position of the triangle and point (x_r, y_r)

1- A translation moves point (x_r, y_r) to the origin

2- Rotation around the origin

3- Inverse translation moves the rotation center back to (x_r, y_r)

- **1** - **Translation** so that the arbitrary point moves to the origin
- **2** - **Rotation** around the origin
- **3** - The **inverse translation** to move the rotation back to its original position

3- Inverse translation

2- θ degrees rotation

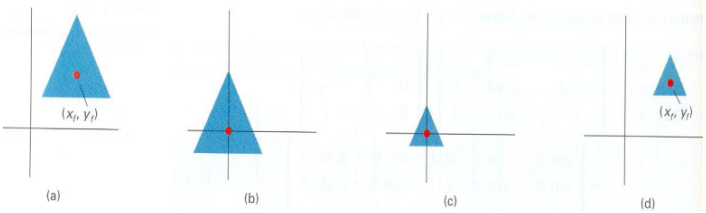
$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

1- Moving to the origin

$$\mathbf{T}(x_r, y_r) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-x_r, -y_r) = \mathbf{R}(x_r, y_r, \theta)$$

Scaling relative to a fixed point



Original position of the triangle and point (x_f, y_f)

1- Move point (x_f, y_f) to the origin

2- Scaling

3- Inverse translation back to (x_f, y_f)

Properties

- Matrix multiplication is **associative**

$$\mathbf{M_3} \,.\, \mathbf{M_2} \,.\, \mathbf{M_1} = (\mathbf{M_3} \,.\, \mathbf{M_2}) \,.\, \mathbf{M_1} = \mathbf{M_3} \,.\, (\mathbf{M_2} \,.\, \mathbf{M_1})$$

- Matrix multiplication is **not commutative**

$$\mathbf{M2} \,.\, \mathbf{M1} = \mathbf{M1} \,.\, \mathbf{M2}$$

/

Efficiency

- A 2D transformation representing a concatenation of transformations can be represented as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} rs_{xx} & rs_{xy} & trs_x \\ rs_{yx} & rs_{yy} & trs_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

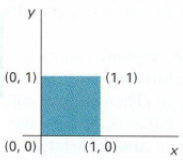
multiplicative terms corresponding to rotations and scalings

terms corresponding to displacement distances, or rotation center or fixed-point for scaling

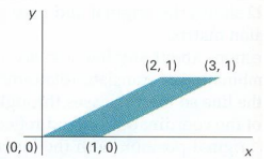
Shearing

$$x' = x + sh_x \cdot y, \qquad y' = y$$

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



unit square

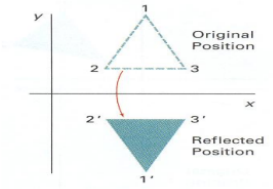


polygon resulting from the XX'-shearing

Symmetry

- To carry out a symmetry relative to the XX' axis *(y=0)* multiply the Y-ccordinates by -1

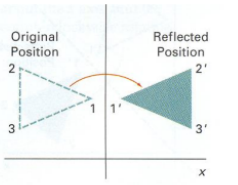
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Symmetry relative to the XX' axis

- To carry out a symmetry relative to the YY' axis *(x=0)* multiply the X-ccordinates by -1

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



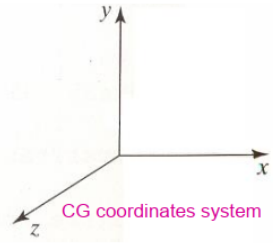
Symmetry relative to the YY' axis

4 - Transformações 3D

- 3D transformation are a generalization of the known 2D transformations
- A point defined in the 3D space if represented, in **homogeneous coordinates**, by a column-vector with 4 elements

$$P = (x,y,z)$$

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



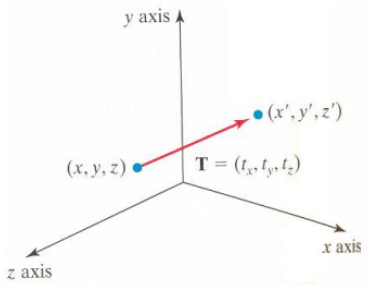
CG coordinates system

Translation

$$x' = x + t_x \qquad y' = y + t_y \qquad z' = z + t_z$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

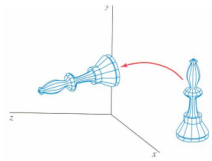
$$\mathbf{P'} = \mathbf{T} \cdot \mathbf{P}$$



Rotation

- Rotations around one of the **coordinate axes** are simpler

Rotation around XX' Rotation around YY' Rotation around ZZ'

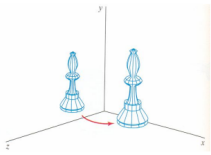


Generalization to 3D

$$y' = y\cos\theta - z\sin\theta$$
$$z' = y\sin\theta + z\cos\theta$$
$$x' = x$$

Matricial representation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

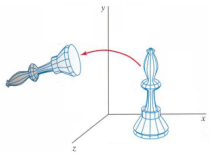


Generalization to 3D

$$z' = z\cos\theta - x\sin\theta$$
$$x' = z\sin\theta + x\cos\theta$$
$$y' = y$$

Matricial representation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Generalization to 3D

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

Matricial representation

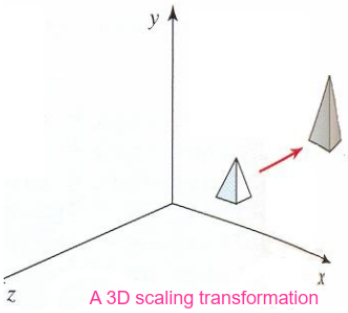
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation around an axis parallel to a coordinate axis

- **Translation** to make the rotation axis coincide with the coordinate axis
- **Rotation** around the coordinate axis
- **Inverse translation** back to the original position of the rotation axis

$$\mathbf{P}' = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T} \cdot \mathbf{P}$$

Scaling



A 3D scaling transformation might change the position of the resulting, transformed object.

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$

$$x' = s_x \cdot x \;,\quad y' = s_y \cdot y \;,\quad z' = s_z \cdot z$$

Scaling relative to a fixed point (Xp, Yp, Zp)

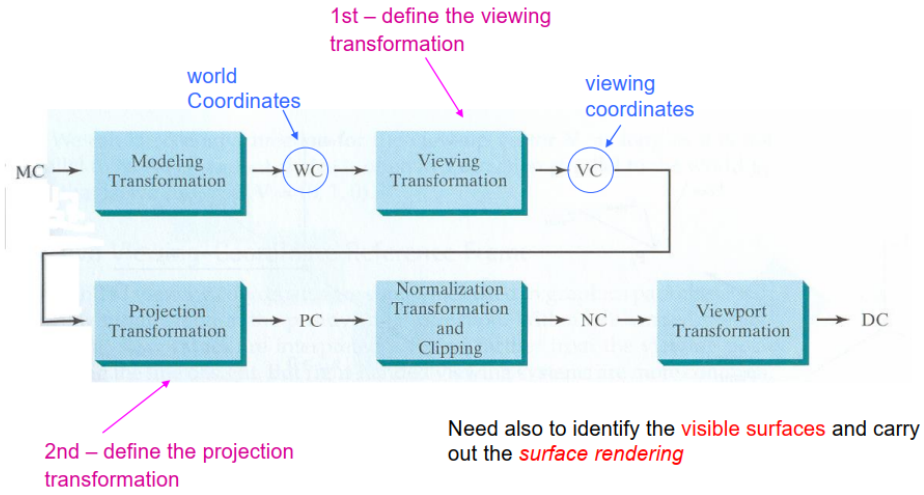
- **Translation** of the fixed point to the coordinates origin
- **Scaling**
- **Inverse translation** of the fixed point back to its original position

$$\mathbf{T}(x_f, y_f, z_f) \cdot \mathbf{S}(s_x, s_y, s_z) \cdot \mathbf{T}(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

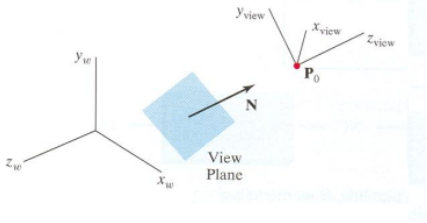
5 - Visualização 3D

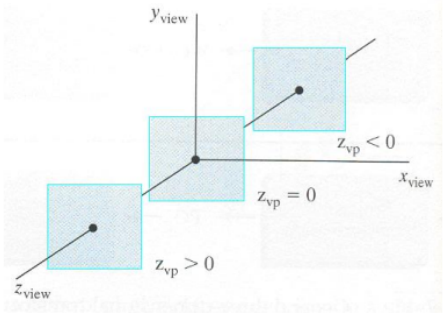
- The process of obtaining a 2D image representing a 3D scene is analogous to photographing
- **Position -** analogous to camera position, depending on the required view
- **Orientation -** analogous to camera orientation

Viewing Pipeline



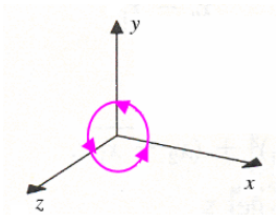
- The **viewing plane** is defined as orthogonal to **Zview**
- The **orientation** of the viewing plane (and the positive direction for **Zview**) is defined by a **normal vector N**
- An additional parameter defined the **position of the viewing plane Zvp** on the **Zview** axes





- The viewing coordinates system is defined as **right-handed**
- In a **right handed** coordinates system, and when looking at the origin from a point on one of the positive semi-axis, CCW 90º positive rotation angles transform a positive semi-axis into another

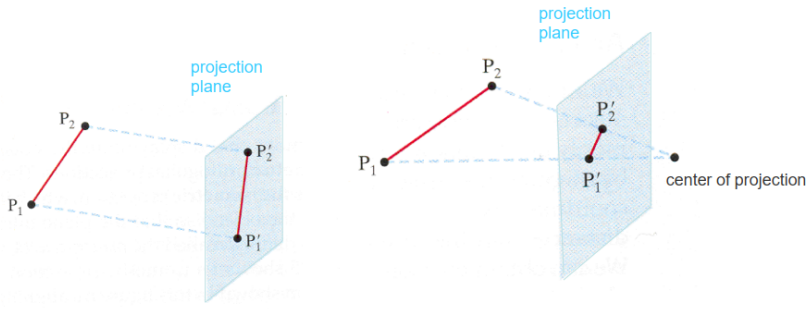
rotation axis	direction of positive rotation
x	y to z
y	z to x
z	x to y



- Maintaining the viewpoint and varying the direction of N, can show models positioned **around the viewpoint** and making up the scene
- Keeping the direction of N and displacing the viewpoint we obtain a **panning** effect

Projections

Parallel projection vs Perspective projection



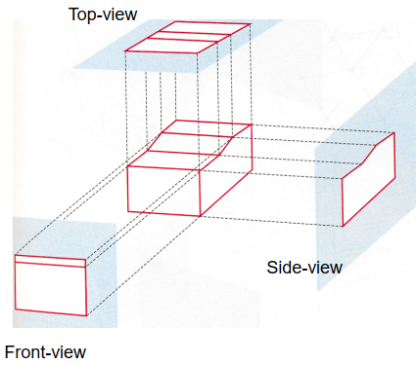
For parallel projections, the projector straight-lines are parallel, i.e., converge at an indefinite distance

For perspective projections, the projector straight-lines converge at the projection center

- **Planar geometric projections** are obtained using **projection straight lines** and **planar surfaces**
    - **Perspective projections** generate **more realistic** images. More calculations

Orthogonal Parallel Projections

- The **projectors** are **perpendicular** to the projection plane
- The **projection plane** is **parallel** to a set of the object's faces
- Some **angles**, **lengths** and **areas** can be directly measured



Axonometric Projections

- Orthogonal parallel projections where the projection plane is **not parallel** to a set of the object's faces
- **Isometric -** the direction of viewing is such that the three axes of space appear equally foreshortened
- **Dimetric -** the direction of viewing is such that two of the three axes of space appear equally foreshortened
- **Trimetric -** the direction of viewing is such that all of the three axes of space appear unequally foreshortened

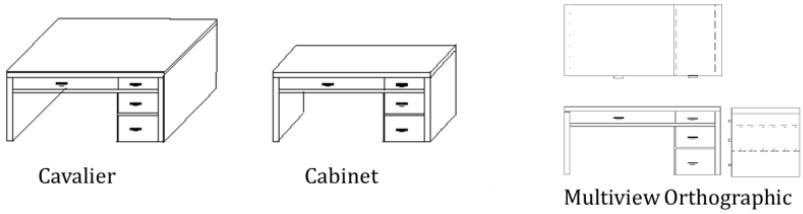
---

Oblique Parallel Projections

- The projections are **oblique** regarding the projection plane
- **Cavalier Projection**
    - Length of the cube's edges is preserved

- Does not look realistic

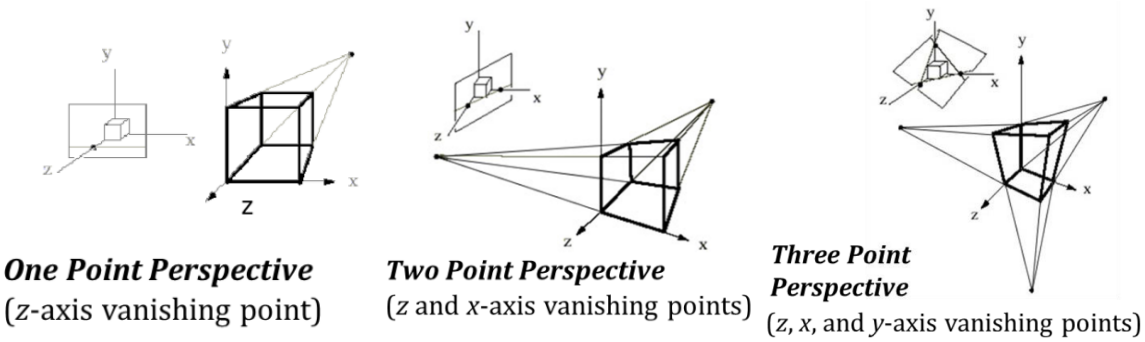
- **Cabinet Projection**
  - Depth of the cube is represented with a **0.5 scale factor**
  - Looks more realistic



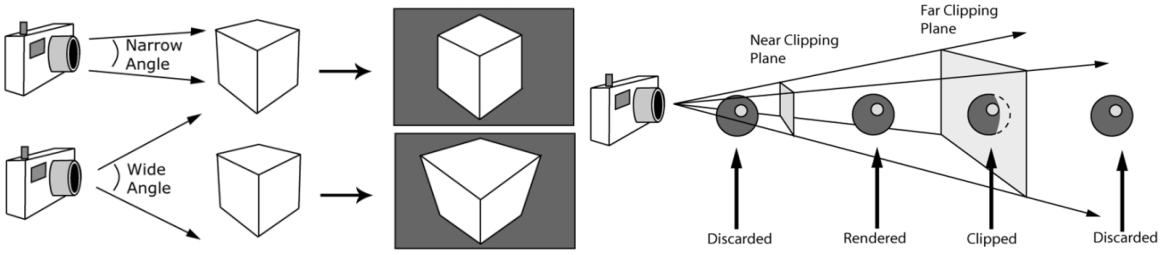
Perspective Projection

- The projections of straight-line segments with the **same length**, but located at different distances form the projection plane, are projected with **different lengths**

Perspective with Vanishing Points



View Angle Clipping Planes



The Mathematics of Planar Projections

- A projection can be achieved through matrix **multiplication**, using a **(4 x 4) projection matrix** in **homogeneous coordinates**
- The **projection matrix** can be **concatenated** with the **model-view matrix** to carry out any modeling transformations before the actual projection

Perspective projection with projection plane at z = d and center of projection at (0, 0, 0)

$P\ (x, y, z)$ – original point
$P_p\ (x_p, y_p, z_p)$ – projected point

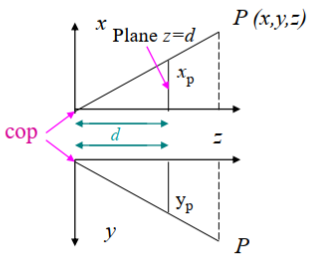
Distance ratios:

$x_p/d = x/z \qquad y_p/d = y/z$

Multiplying by d:

$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d}$$

$$y_p = \frac{d \cdot y}{z} = \frac{y}{z/d}$$



All z values are possible except $z=0$

The projection matrix in homogeneous coordinates:

$$M_{pers} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \longrightarrow P_p = M_{pers} \cdot P$$

Dividing by z implies that objects further away appear smaller

Perspective projection with plane at z = 0 and center of projection at (0, 0 ,-d)

$P\ (x, y, z)$ – original point
$P_p\ (x_p, y_p, z_p)$ – projected point

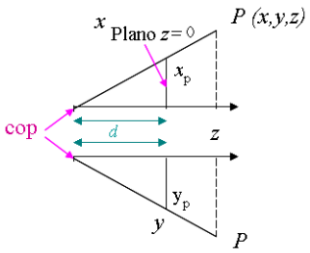
Distance rations:

$x_p/d = x/(z+d) \qquad y_p/d = y/(z+d)$

Multiplying by d:

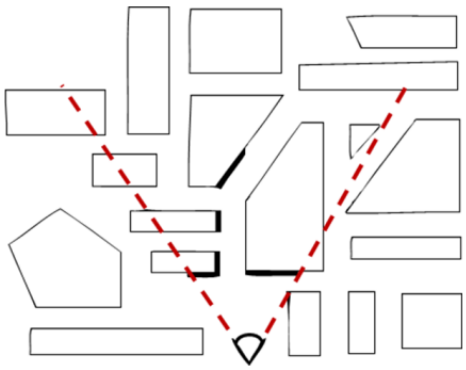
$$x_p = \frac{d \cdot x}{z+d} = \frac{x}{z/d+1}$$

$$y_p = \frac{d \cdot y}{z+d} = \frac{y}{z/d+1}$$



$$M'_{pers} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}$$

6 - Determinação da Visibilidade de Superfície



- For each object compute
  - The **visible edges and surfaces**

Clipping vs Occlusion

- **Clipping** against the **view volume**
  - It is done at **object-level**
- **Occlusion** / Hidden-Surface Removal
  - It is done at **scene-level**

Possible Approaches

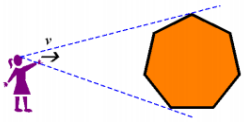
- **Object-precision algorithms**

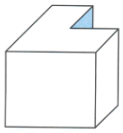
    - Analyze / compare objects or parts of objects to determine which surfaces / faces / edges are fully or partially visible
    - **Back-Face Culling**
- **Image-precision algorithms**
    - Determine visibility for every pixel in the viewing plane. Work in 3D to get / compare depth values
    - **Z-Buffer**, **A-Buffer**, **Ray-Casting**

Determine which **models** or **parts of models** are **visible** from the chosen **viewpoint** and **projection type**

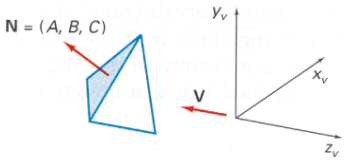
Back-Face Culling

- Sufficient for **a single convex polyhedron** which is not sectioned by clipping
- Not sufficient for **concave polyhedral** or when there are **two or more models** in front of each other



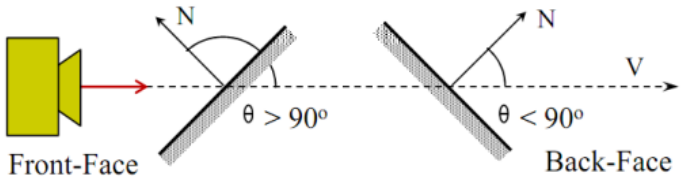


Concave model with a partially visible face



When looking at the negative ZZ' semi-axis, a polygon is a back-face if C ≤ 0

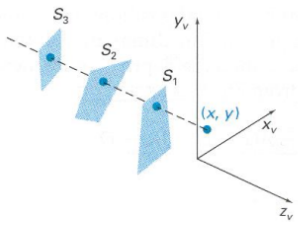
- **For each face**, compute the **angle** between
    - The **normal vector** to the face
    - The **viewing direction**, defined by the viewpoint



- Compute the **scalar product**
    - **Reject** a face if **N . V > 0**
- **Simplification** when **V = (0, 0, -1)**

Depth-Buffer (Z-buffer)

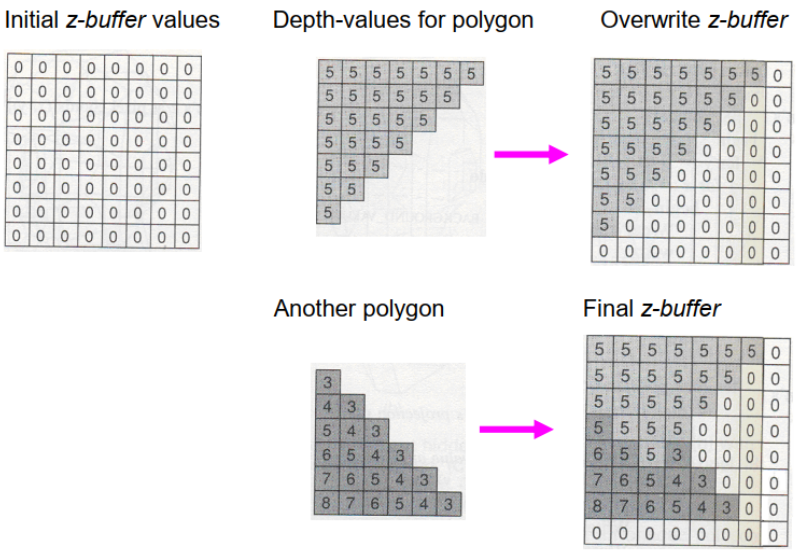
- Works in **image-space**
- Compares the **depth** of each surface relative to each **pixel** in the **view plane**
- Fast and easy to implement for **planar surfaces**
- Needs a **depth-buffer** in addition to the **frame-buffer**



S1 is visible at pixel *(x,y)*, since it is closer to the viewplane

Z-Buffer Algorithm

- Draw **every polygon** that can't be rejected trivially
- If **a piece** (one or more pixels) of a polygon that is **closer to the front** is found
    - **Paint** over whatever was behind it



Advantages

- Easy, no need to previously sort the various surfaces. Fast

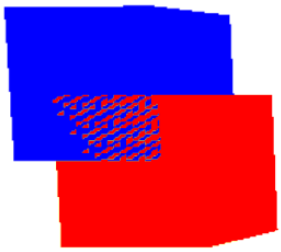
Disadvantages

- Need for **additional memory**
- Depth **precision** problems / limitations
- It can only handle **opaque surfaces**

Z-Fighting

Z-fighting occurs when two primitives have similar values in the z-buffer

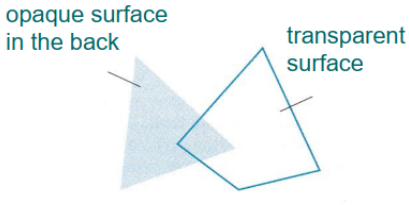
- Coplanar polygons (two polygons that occupy the same space)
- One is arbitrarily chosen over the other, but z varies across the polygons and binning will cause artifacts
- Behavior is deterministic: the same camera position gives the same z-fighting pattern



Two intersecting cubes

A-Buffer

- Anti-aliased area-average, **accumulation buffer**
- An **extension of the depth-buffer**, allows to take into account more than one surface (handling **transparent surfaces**)
- It also allows for **lesser aliasing** of model edges



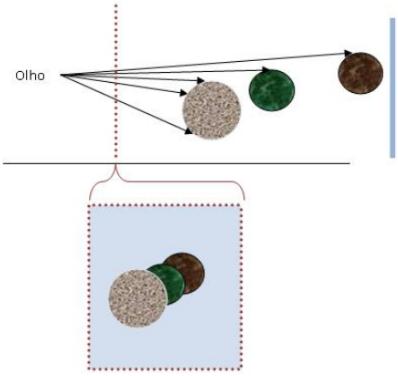
To see an opaque surface
through a transparent surface
requires the accumulation of the
contributions from both surfaces

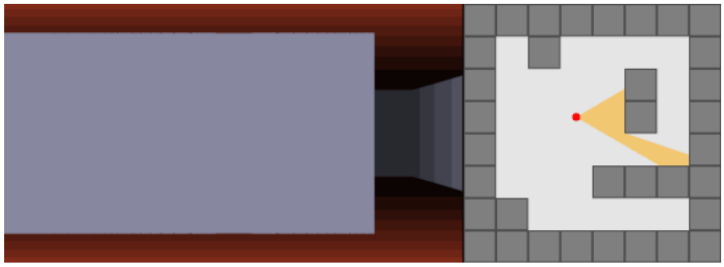
Depth-sorting (Painter's Algorithm)

- **Image-space** and **object-space** operations
- Main steps
  - **Sort** the surfaces by their **depth** relative to the viewplane
  - **Back-to-front projection** of the sorted surfaces
- **Subdivide** the surfaces or change their order if **superpositions** occur
- **No** need for a **depth-buffer**
- Renders distant surface parts that, in the end, will **not be visible**

Ray-Casting

- **Image-space** method
- Which models are intersected by a **ray cast from the viewpoint** that passes through the **center of a pixel**
- If models are opaque, the **intersection point closer to the viewpoint** determines the color for the corresponding pixel





7 - Métodos de Iluminação e Sombreamento

Lighting or Illumination

- The process of **computing** the **intensity** and **color** of a sample **point** in a scene as seen by a **viewer**
- It is a function of the **geometry** of scene and of **material** properties

Shading

- The process of **interpolation** of **color** at points **in-between** those with **known lighting** or illumination

- In **ray-tracing** only do lighting for samples. Based of pixels. **No shading rule**

GPU

- Lighting is usually calculated by a **vertex shader**
- While shading is done by a **fragment or pixel shader**

Global Illumination

- **Total** illumination = **Direct** illumination + **Indirect** illumination

Light Transport Simulation

- Evaluate illumination with enough **samples** to produce final images **without any guessing / shading**
- High quality renderers. **FX movies**
- Done in the **CPU**
- Many simulations use **stochastic sampling**

Polygon Rendering

- Evaluate illumination at **several samples**
- **Shade** in between to produce **pixels** in the final image
- Real-time applications such as **computer games**
- Done in the **GPU**

Realistic images by

- Using **perspective projections** of the scene models
- Applying **natural illumination effects** on the visible surfaces

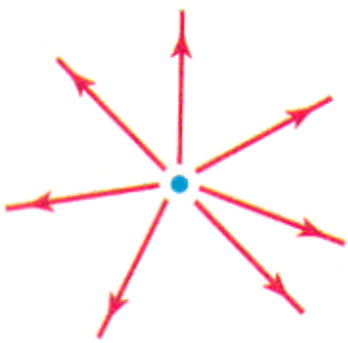
Natural illumination effects

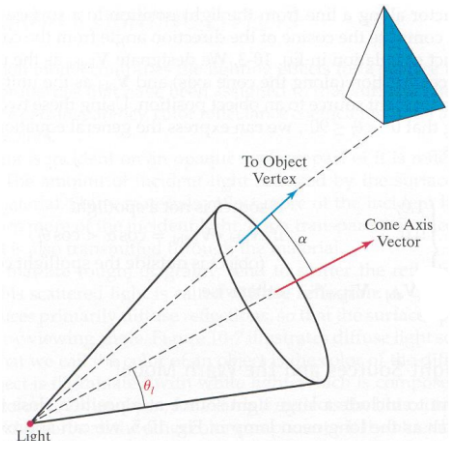
- An **illumination model** allows computing the **color** to be assigned to each visible **surface point**
- A **surface rendering method** that applies an illumination model and assigns a **color to every pixel**
- **Illumination models** are often an approximation to the **Laws of Physics** that describe the **interaction light to surface**

Light Sources

- Objects **radiating light** and contributing to the illumination of a scene's objects

Isotropic Light Source Directional Light Source





- A directional light source is defined by a **direction** and an **emission angle**

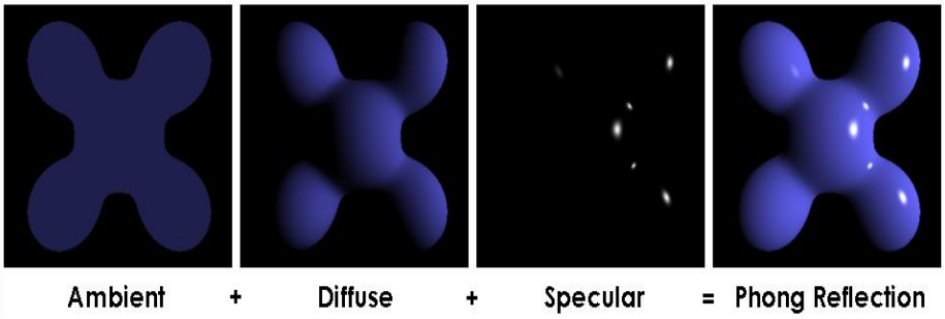
Surface Features

- An illumination model takes into account
  - **Reflection coefficients** for each color
  - Degree of **transparency**
  - **Texture** parameters
- When light is incident on an **opaque surface**
  - Part is **absorbed**. Part is **reflected**
  - **Shiny** surfaces reflect **more** light
  - **Mate / Dull** surfaces reflect **less** light
  - **Transparent** surfaces transmit some light
  - **Rough** surfaces tend to spread the reflected light in all directions (**Diffuse reflection**)
  - **Smooth** surfaces reflect more light in particular directions (**Specular reflection**)
- **Ambient illumination -** A surface **might not be directly illuminated** and still be **visible**, due to light reflected by other objects in the scene

---

The amount of light reflected by a surface is the **sum of all contributions** from the light sources and the ambient illumination

Phong Reflection Model



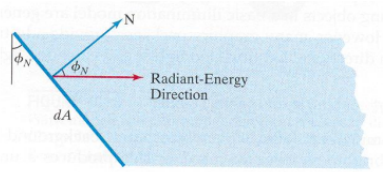
Ambient Illumination

- Included as a **constant value** for the whole scene which entails a **uniform illumination** for all objects
- The **reflections** produced by the surfaces are
  - Independent of **viewing direction**
  - Independent of **surface orientation**
  - Just depend on the **optical properties** of the surface

Diffuse Reflection

- **Incident light** is spread with **equal intensity in all directions**, regardless of the viewing direction
- Surfaces with that feature are called **Lambertian reflectors** or **ideal diffuse reflectors**
- **Lambert's Law**

$$Intensity = \frac{Radiant\text{-}Energy\ per\ time\ unit}{Projected\ Area}$$

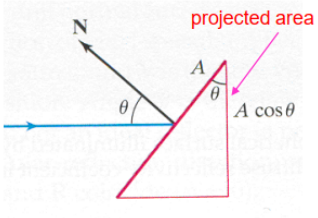


- The amount of incident light depends on the **surface orientation** regarding the **direction of the light source**

  • θ is the incidence angle (between the surface normal and the light direction)

  • Given a light source I_l, the amount of diffusely reflected light by a surface is:

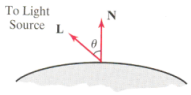
$$I_{l,\text{diff}} = k_d I_l \cos\theta$$



Diffuse Reflection + Ambient Lighting

N → surface normal
L → light source direction

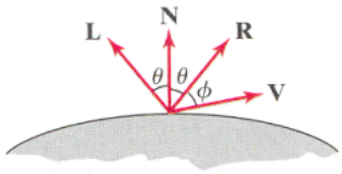
$$I_{l,\text{diff}} = \begin{cases} k_d I_l (\mathbf{N} \cdot \mathbf{L}), & \text{se } \mathbf{N} \cdot \mathbf{L} > 0 \\ 0.0, & \text{se } \mathbf{N} \cdot \mathbf{L} \leq 0 \end{cases}$$



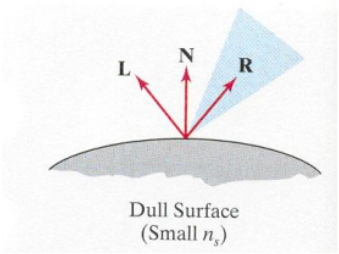
$$I_{\text{diff}} = \begin{cases} k_a I_a + k_d I_l (\mathbf{N} \cdot \mathbf{L}), & \text{se } \mathbf{N} \cdot \mathbf{L} > 0 \\ k_a I_a, & \text{se } \mathbf{N} \cdot \mathbf{L} \leq 0 \end{cases}$$

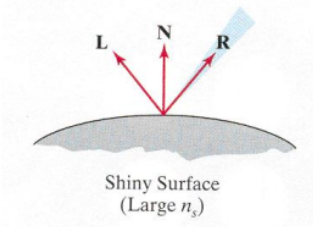
Specular Reflection

- The shinier areas, **specular reflections** or **highlights**, result from the reflection of most light around concentrated areas



- The **specular reflection angle** is equal to the incidence angle
- **R** is the **unit vector** defining the ideal specular reflection direction
- **V** is the **unit vector** defining the viewing direction
- An **ideal reflector** reflects light only in the specular reflection direction
- Objects that are not ideal reflectors produce specular reflections in a **finite set of directions** around **R**





- The empirical **Phong specular reflection model** sets the intensity of the specular reflections

$$\cos^{n_s}\phi$$

$$I_{l,\text{spec}} = W(\theta) I_l \cos^{n_s}\phi$$

- W(θ) is the **specular reflection coefficient**

- The value of **cos ϕ** can be computed by the **scalar product** of unit vectors **V . R**

- There are not specular reflections, whenever
  - The light source is behind the surface
  - **V** and **L** are on the same side of vector **N**

cos Φ

$$I_{l,\text{spec}} = \begin{cases} k_s I_l (\mathbf{V} \cdot \mathbf{R})^{n_s}, & \text{if } \mathbf{V} \cdot \mathbf{R} > 0 \quad \textit{and} \quad \mathbf{N} \cdot \mathbf{L} > 0 \\ 0.0, & \text{if } \mathbf{V} \cdot \mathbf{R} < 0 \quad \textit{or} \quad \mathbf{N} \cdot \mathbf{L} \leq 0 \end{cases}$$

R and V on opposite sides — Light source behind the surface

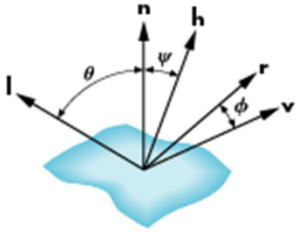
Single Light Source Multiple Light Sources

$$I = k_a I_a + k_d I_l (\mathbf{N} \cdot \mathbf{L}) + k_s I_l (\mathbf{N} \cdot \mathbf{H})^{n_s}$$

$$I = k_a I_a + \sum_{l=1}^{n} I_l [k_d (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{N} \cdot \mathbf{H})^{n_s}]$$

The Halfway Vector

- **H** is the unit vector "halfway" between **I** and **V**



The BRDF

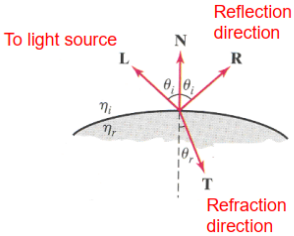
- Light arriving at a surface can **scatter** in many directions
  - The direction of scattering is determined by the **material**
  - Intensity at a given outgoing direction is dependent on **incoming direction** and material properties
- Model of reflectance is called the **bidirectional reflectance distribution function (BRDF)**

Transparency

- A **transparent** object allows seeing other objects behind it
- A **translucent** object transmits light but scatters it in all directions
- **Light Refraction** - light rays change direction
- The **refraction angle** varies with
  - The media **refraction indices** Ni Nr
  - The **incidence angle** θi

Snell's Law

$$\sin\theta_r = \frac{\eta_i}{\eta_r} \sin\theta_i$$



To compute T (unit vector for the refraction direction)

$$\mathbf{T} = \left(\frac{\eta_i}{\eta_r}\cos\theta_i - \cos\theta_r\right)\mathbf{N} - \frac{\eta_i}{\eta_r}\mathbf{L}$$

N – surface unit normal vector

L – unit vector towards the light source

- It is possible to combine light transmitted through a transparent surface with light reflected by that surface

- It is possible to combine light transmitted through a transparent surface with light reflected byt that surface:

$$I = (1 - k_t) I_{\text{refl}} + k_t I_{\text{trans}}$$

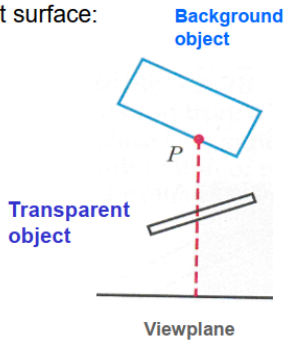
k_t - transparency coefficient [0, 1]

1 – fully transparent

0 – totally opaque

- Can also define the opacity coefficient:

$$(1 - k_t)$$



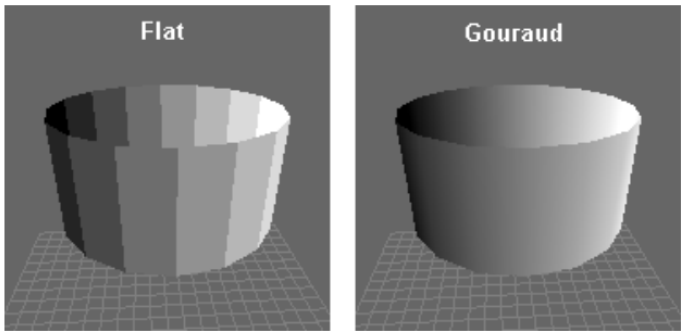
Shadows

- Identify the scene's **polygonal faces** that are **not visible** from the location of each light source

Shading

Flat-shading

- Uses the illumination model to compute RGB color values at a chosen point for each polygon
- Assigns the **same color** to **all pixels** in the projection of each polygon
- **Simple** and **fast**. Appropriate for simple cases and useful to obtain the general appearance of a curved object. Objects seem "**blocky**"
- **Flat shading is adequate when**
  - The polygon is a face of a **polyhedron**
  - All **light sources** are very **far-away**
  - The **viewpoint** is very **far-away**



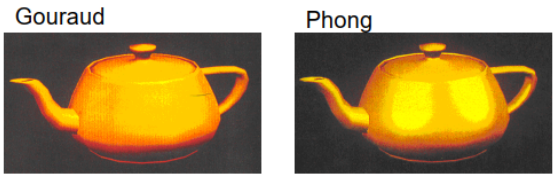
Gouraud Shading

- For each triangle / polygon
  - Apply the illumination model at **each vertex**
  - **Interpolate** color to shade each pixel
- Better than flat-shading. Problems with **highlights**. **Mach-effect**
- Computes **color intensity values** at **mesh vertices** and linearly **interpolates color** along the mesh polygons
- **3 Steps**
  - Compute the **average normal vector** for each polygon vertex
  - **Apply the illumination model** at each vertex to compute color intensity values

- **Linearly interpolate color intensity values** along the projected polygonal area

Phong Shading

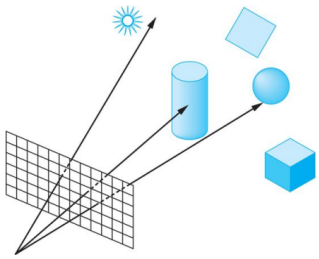
- For each triangle / polygon
  - **Interpolate normal vectors** across rasterized polygons
- Linearly **interpolates normal vector** (instead of color intensity values) and **repeatedly applies the illumination model**
- **3 Steps**
  - Compute the **average normal vector** for each polygon vertex
  - **Linearly interpolate normal vectors** along the projected polygonal area
  - **Apply the illumination model** along the scan-lines to compute color intensity values



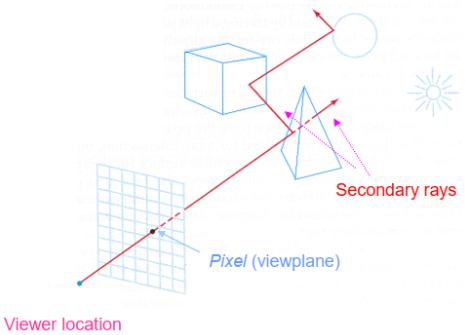
Ray-Tracing

- A finite **back-mapping of rays** from camera (eye) through each sample (**pixel**) to objects in scene
- Each **pixel** represents either
  - A ray intersection with an object / light source in scene
  - No intersection
- A ray traced scene is a **virtual photo** comprised of many samples on film plane

Ray Casting



- In addition, might cast **secondary rays**. Whenever there are secondary rays -» **Ray Tracing**



Fundamentals

- Generate **primary ray**
  - Shoots rays from eye through sample points on film plane

- Ray object **intersection**
  - Find first object in scene that ray intersects with
- Calculate Lighting (**color**)
  - Use illumination model to determine **direct** contribution from light sources (**light rays**)
  - Reflective objects **recursively** generate secondary rays (**indirect**) that also contribute to color
- **Sum** of contributions determines color of sample point
- RT uses **specular reflection** rays. **No diffuse reflection** rays
- **Limited approximation** to global illumination

8 - Texturas

Texture Mapping

- Implemented in hardware on every **GPU**
- Paste the texture on a surface to **add detail** without adding more triangles
- Uses Images to **fill inside triangles**

Bump Mapping

- Emulates altering **normal vectors** during the rendering process

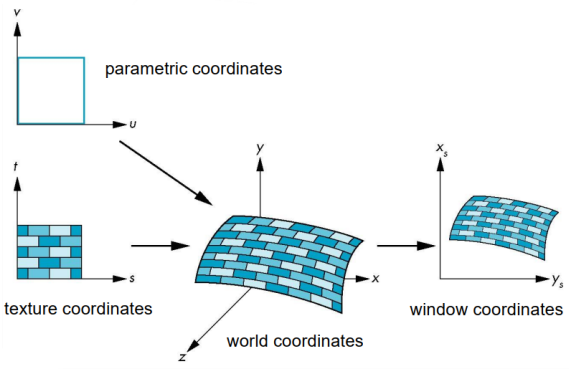
Environment Mapping

- Uses a picture of the environment for texture maps
- Allows **simulation** of highly **specular surfaces**

Mapping techniques are implemented at the **end** of the rendering **pipeline**. To map an image to a surface there are 3 or 4 coordinate systems involved

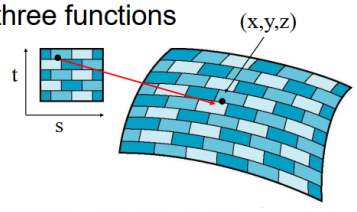
Coordinate Systems

- **Parametric Coordinates -** May be used to model **surfaces**
- **Texture Coordinates -** Used to identify points in the **image** to be mapped
- **Object or World Coordinates -** Where mapping takes place
- **Window Coordinates -** Where the final image is really produced



Appear to need three functions

$$x = x(s,t)$$
$$y = y(s,t)$$
$$z = z(s,t)$$

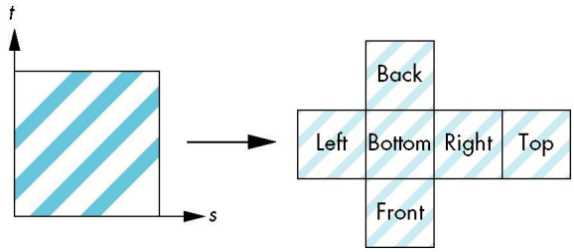


Backward Mapping

- Given a **pixel**, we want to know o which **point on an object** it corresponds
- Given a **point on an object**, we want to know to which **point in the texture** it corresponds

Box Mapping

- Easy to use with simple orthographic projection

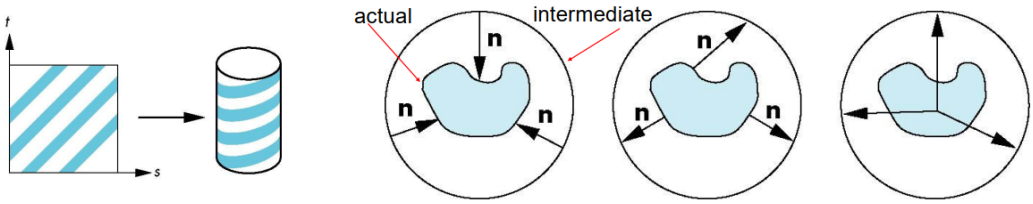


Two-part Mapping

- One solution to the mapping problem is to **first** map the texture to a simple **intermediate surface**

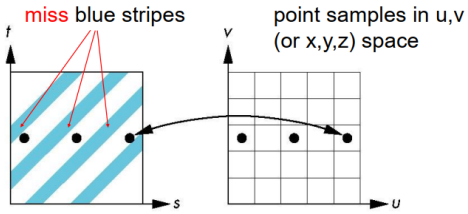
Second Mapping

- Map from intermediate object to actual object
  - Normals from intermediate to actual
  - Normals from actual to intermediate
  - Vectors from center of intermediate



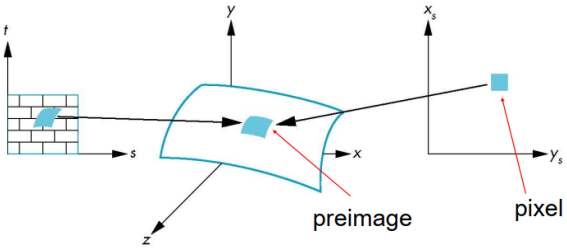
Aliasing

- Point sampling of the texture can lead to **aliasing errors**



Area Averaging

- A better but slower option is to use **area averaging**



Basic Strategy

- Three steps to applying a texture
  - Specify the **texture**

        - Read or generate image
        - Assign to texture
        - Enable texturing
    - Assign **texture coordinates** to **vertices**
        - Proper mapping function is left to application
    - Specify texture **parameters**
        - Wrapping, filtering

WebGL pipeline

- **Geometry** and **images** flow through **separate pipelines**

Specifying a texture image

- Define a texture image from an **array of texels**
- Use an image in a **standard format** such as JPEG

Magnification - **more than one pixel can cover a** textel**

Minification - more than one **texel** can cover a pixel

Mipmapping - allows for prefiltered texture maps of **decreasing resolutions**

Applying Textures

- A texture **fully determines color**
- A texture is **modulated** with a computed **color**
- A texture is blended with an **environment color**

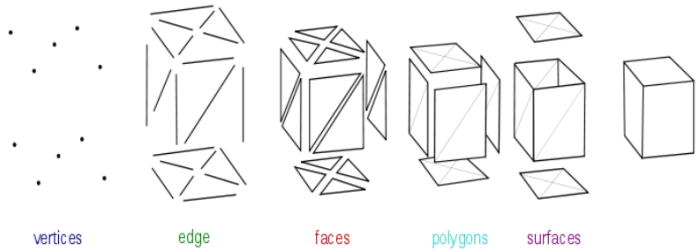
9 - Modelação Geométrica com Malhas Poligonais

Geometric Modeling

- A **geometric model** describes the **shape** of an object
- **Main Areas**
    - Curve and surface modeling
    - Solid Modeling
    - Volume Modeling
- **Simplest Models**
    - **Curves** - Polygonal lines
    - **Surfaces** - Polygonal meshes

Polygonal Meshes

- Surface is defined as a collection of **neighboring faces**
- Vertices, edges, faces
- Euler formula for closed surfaces
    - V + F - E = 2



- Collection of neighboring **vertices**, **edges** and **polygons**. Usually **triangles**
- **Vertex -** Shared by, at least, 2 edges
- **Edge -** Connects 2 vertices. Shared by 2 polygons
- **Polygon -** Sequence of, at least, 3 vertices

- **Surface -** Defined as a set of adjacent faces
- **Geometric Information** to be stored -» Vertex Coordinates
- **Topological Information** to be stored -» How are faces and edges arranged
- **Additional Properties -** Texture Coordinates

Polyhedral Models

- The same polyhedral model might be represented by **different polygonal meshes**
- Degrees of freedom
  - **Number** of mesh vertices
  - **Distribution** of mesh vertices
  - **Arrangement** of edges / polygons

Curved Surfaces

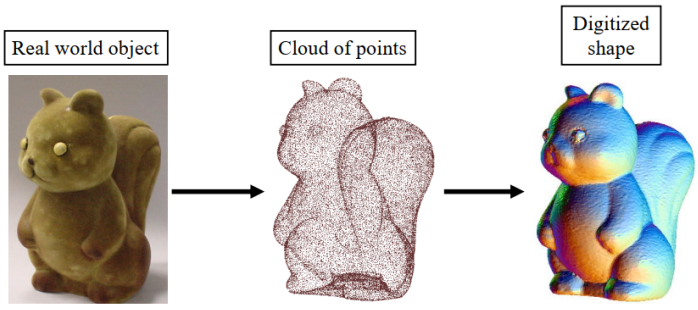
- Representing the shape of a curved surface is an **approximation** process

---

Refinement - Increase surface smoothness

Decimation - Decrease the number of vertices / polygons

Digitizing



Topological Information

- **Vertex -** Regular or Singular
- **Edge**
  - Border Edge -» 1 incident face
  - Regular Edge -» 2 incident faces
  - Singular Edge -» 3 or more incident faces
- **Loop -** Ordered edge sequence

- **Face -** Limited by a set of disjoint edge sequences
- **Shell -** Set of connected faces

Valid vs Non-Valid Models

2 - Manifold Model

- Any point has a "disk" neighborhood
- No singular vertices
- No singular edges

Euler Formula

- Allows checking the consistency of the topological information
- **V + F - E = 2**
- When to apply?
    - Model has a closed, orientable surface
    - Each face is limited by a single edge loop
    - No through-holes
    - Nor cavities

Euler-Poincaré Formula

- **V + F - E - (L - F) - 2 (S - G) = 0**
- **L -** Number of loops
- **S -** Number of shells
- **G -** Genus - number of handles

Computational Representation

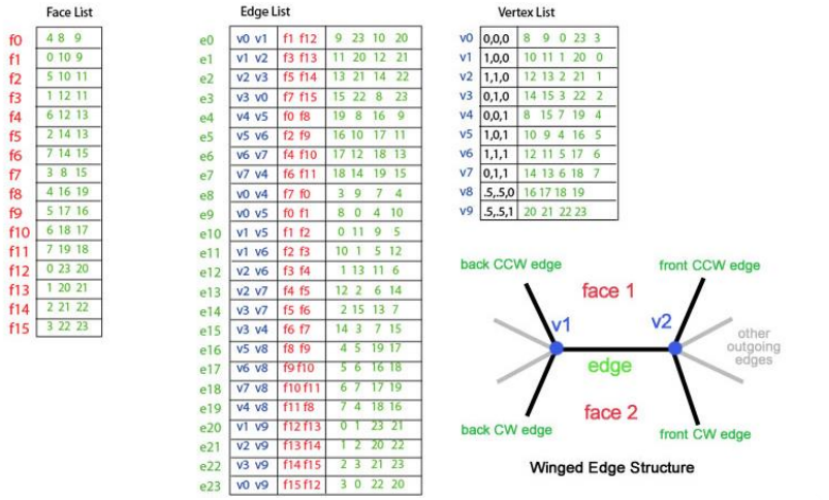
Detached / Isolated Polygons List - Each polygon is represented by the ordered list of its vertices coordinates - **Inefficient -** Memory space

STL File Format - Stereolithography File Format. Mesh defined by T detached triangles. Each triangle defined by 3 vertices. Unit normal vector for each triangle

Vertices List - Store just once the coordinates of each vertex. Each polygon is described by its vertices sequence - **Inefficient -** Hard to detect which polygons share a given edge

The Winged-Edge data structure

- Explicit representation of vertices, edges and faces
- Dynamic mesh modification

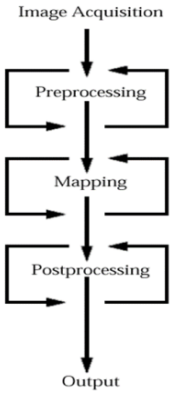


10 - Introdução ao Processamento de Imagem

Digital Image Acquisition

- Continuous light distribution is **spatially sampled**
- Still image created by **time sampling** that discrete distribution
- Resulting values are **quantized** to a finite set of numerical values

Image Processing Pipeline



- **Acquisition -** obtain images
- **Preprocessing -** image **crop**, **mask**, **filter**. etc
- **Mapping -** image **transformations** or image **composition**
- **Postprocessing -** texturizing, **color remapping**, etc
- **Output -** archiving, printing or displaying on a screen
- Stages may be skipped

Synthesis - images created by a computer

Capture - images from the real world

Preprocessing

- Fit an image to a given **tone**, **size**, **shape**, etc.
- Match an image to a desired feature or to other images
- Make a set of **dissimilar images** appear similar
- Make **similar parts** of an image appear dissimilar

- **Adjusting** color or grayscale
- **Cropping**
- **Masking -** cutting out part of an image
- **Blurring / Sharpening**
- **Edge** detection / enhancement
- **Filtering** / Anti-aliasing
- **Super-Sampling** / Sub-Sampling

Mapping

- Apply **geometrical transformations**
- Combine images - **compositing**

Postprocessing

- Create **global effects** across an entire image or selected area
- **Art** effects
- **Technical** effects

Operate on Pixel Position - Correction of geometric distortion. Distortion of an image into another. Texture mapping

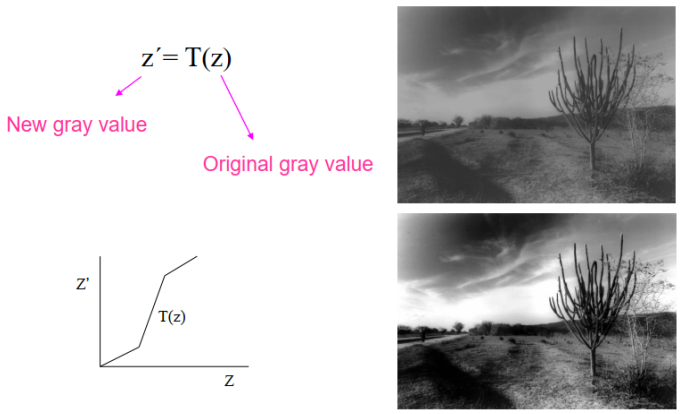
---

Pixel Processing

- Methods for **pixel processing** in the **space-domain**. They operate on pixel **intensity or color values**
- A **transformation** is applied to the **values** associated with **image pixels**
- Such a transformation can be established based on global image features (**histogram**) or considering the **pixel values in the neighborhood** of each image pixel

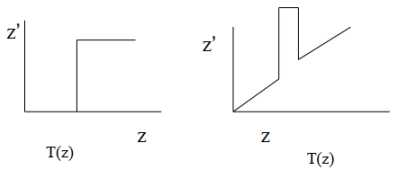
Intensity Transformations

- The same transformation is applied to **all image pixels**
- **Contrast stretching** is one of the mostly used transformations
- For **grayscale** images that do not use the entire **dynamic range** the histogram can be modified



Intensity Level Slicing

- **Highlighting** a contiguous set of intensity values
- **Thresholding** is a particular case and can be used as a segmentation technique



a) Thresholding *b) Highting an intensity band of interest*

Intensity Level to Color

- Transform a grayscale image into a color image
- **Recodification** of the original image
- Use in images that are no photos and depict some kind of data
- **False contours** might be perceived

Pseudo-color vs False-color

- Given an **original image**, with **3 non-RGB** channels, the **false color** transforms that image into an **RGB image**
- **Recodification** of the original image

Histogram Based Processing

- Histogram provides a general idea of the image **gray levels** and allows choosing an adequate processing operation
- Ideally an image histogram should occupy the **entire grayscale**
- **2 stages**
    - Computing and analyzing the **histogram**
    - **Image enhancement** based on histogram features

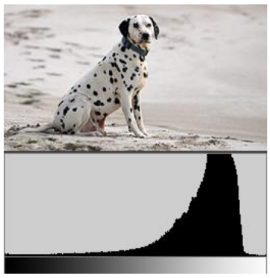
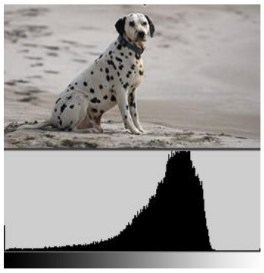
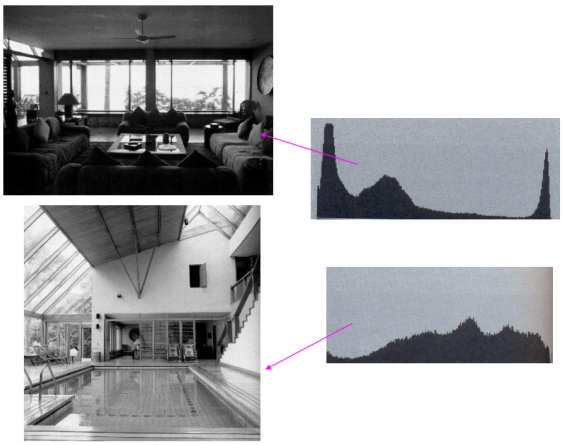


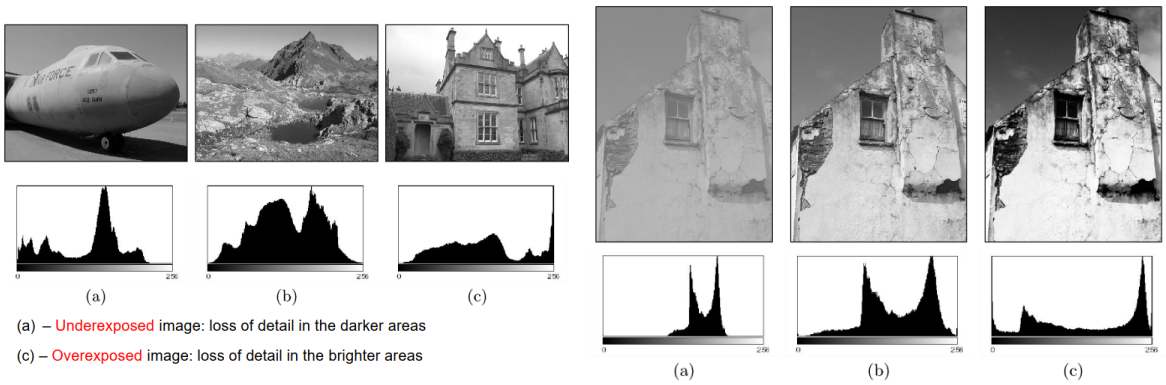
Image and histogram



Underexposed image:
slightly more dark



Exposure Contrast



(a) – Underexposed image: loss of detail in the darker areas

(c) – Overexposed image: loss of detail in the brighter areas

Histogram Equalization

- Extension of **contrast-stretching**
- The image is analyzed and a function T(z) is computed in order to obtain an image with a ~ **equiprobable histogram**

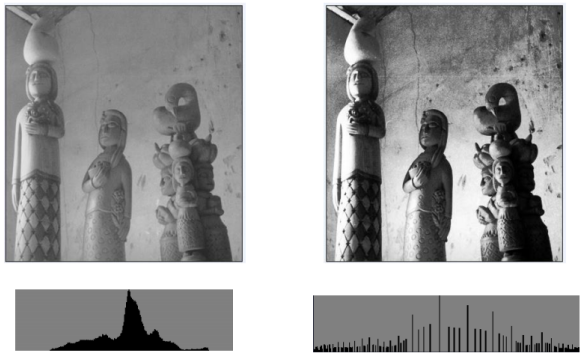


Image Subtraction

- Can be used to **isolate movement** happening between two successive images
- As well as for **image segmentation and enhancement**

Image Smoothing

Average Filter

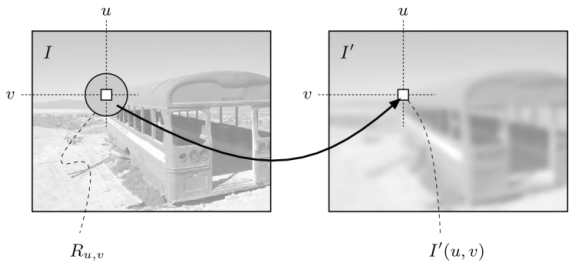
- Also called **neighborhood averaging**, can be used to reduce image noise
- Or as a preprocessing step prior to edge detection
- Simple filtering operation, where the **filter kernel** slides over the original image and the **intensity** of the resulting pixel in the final image is computed as

3 x 3 Averaging Filter
All weights are 1/9

$$\sum_{i=0}^{N-1} W_i \cdot I_i$$

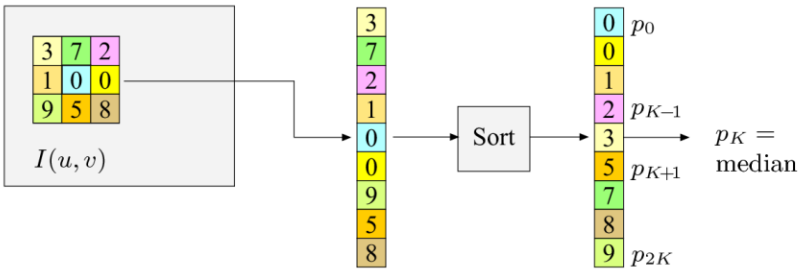
Pixel intensity values in the original image

Filter weights

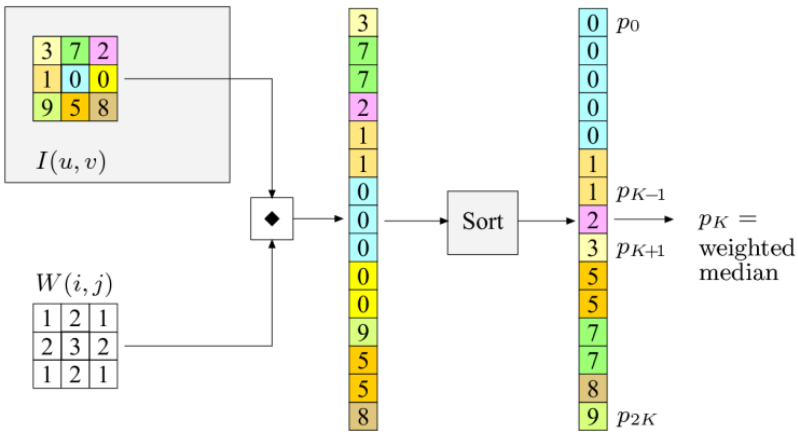


- The averaging filter is a simple **approximation** to the **low-pass filter** in the frequency domain, where, the larger the filter area the lower its cutoff frequency
- Smoothing is carried out as an operation in the **space-domain**

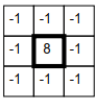
Median Filter



Weighted Median Filter



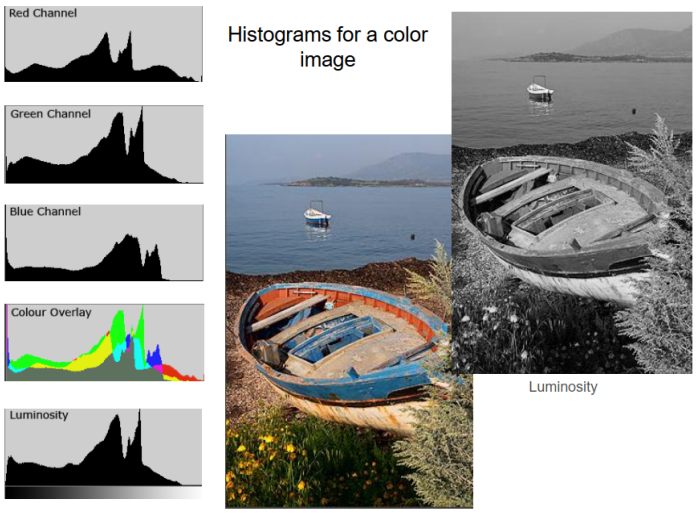
Sharpening



- It is an **approximation** to the **high-pass filter** in the frequency-domain
- Implemented in the **space-domain**
- If all the pixels in the neighborhood have the **same intensity** value, the corresponding pixel in the result image has value **zero**

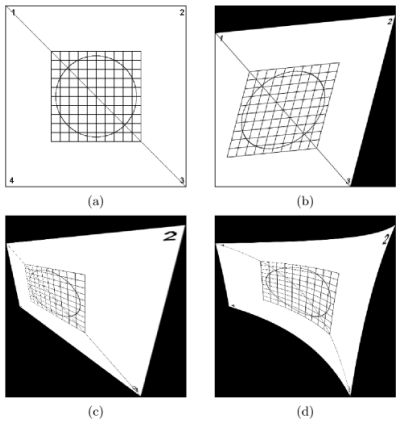
Process Color Images

- Defined using **3 RGB Components**

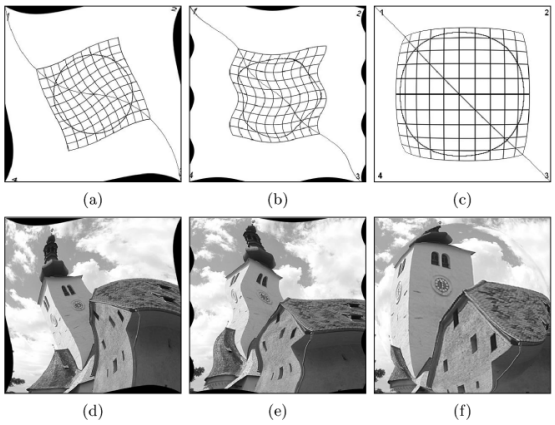


Global Geometric Transformations

Geometrical transformations: affine, projective, bilinear

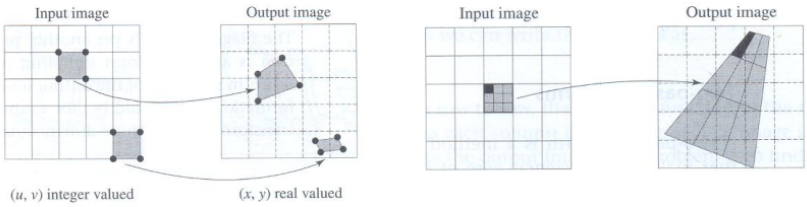


Non-linear transformations: *Twirl*, *Ripple*, *Sphere*



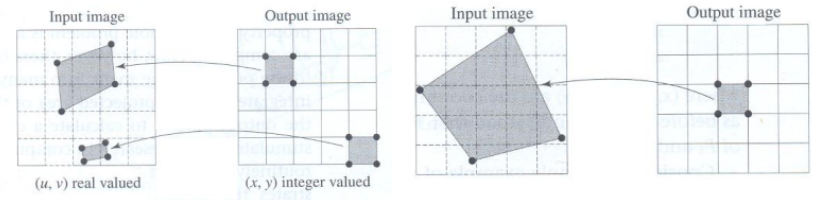
Forward Mapping

- **Gaps** and **superpositions** can happen in the output image
- Computing the **intensity** of the output image pixels is not straightforward
- **Magnification** can induce **blocky** appearance



Inverse Mapping

- All output image pixels are computed
- **Gaps** no longer occur
- If an output pixel is mapped on several input pixels, it is necessary to **integrate** the input pixels values to obtain the value of the output pixel

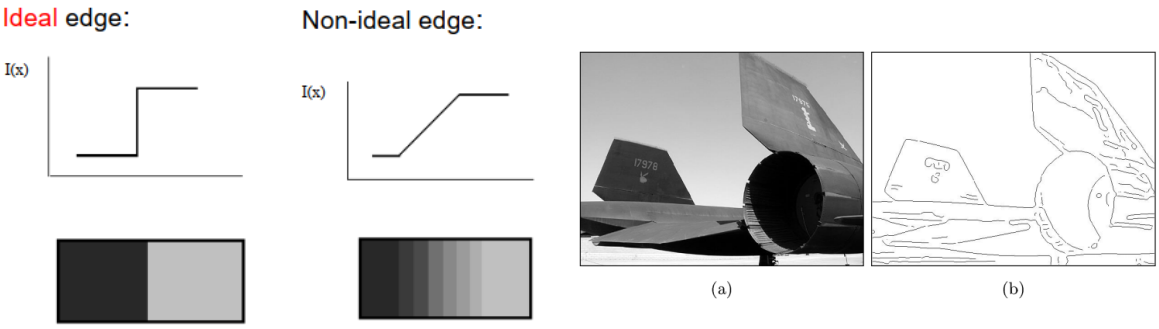


> Exercícios Importantes de Filtros

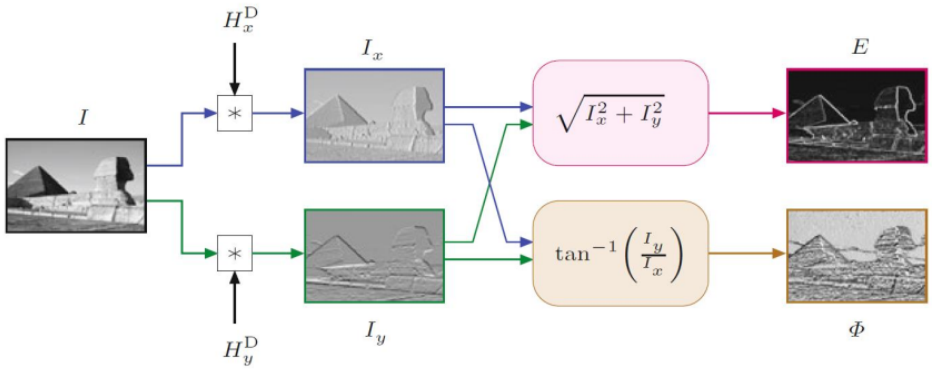
11 - Deteção de Arestas e Segmentação de Imagens

Edge Detection

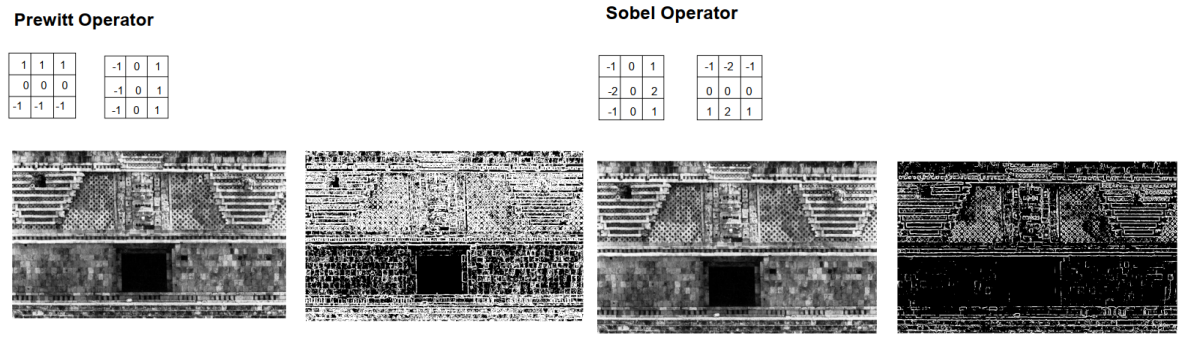
- Edges help reveal / understand **contents** of an image
- **Affected** by
  - **Noise**
  - **Non-ideal edges**



- Detected in **2 steps**
  - Applying a **kernel** (derivative approximation)
  - Aggregating detected pixels (**edgels**) into edgels
- Using **derivatives** along an **image row**
  - The **1st derivative** is
    - **> 0** or **< 0**, depending on the **variation of I (x, y)**
    - **= 0**, in areas of **constant I(x, y)**
  - The **2nd derivative** goes through **0** at "positive" and "negative" edges
- To compute **derivatives** of a discrete image f(x, y), one option is to compute **finite differences** and apply the corresponding **kernel**



Prewitt and Sobel Operators (Edges with a pre-defined Orientation)



> Roberts na diagonal

Edge Detectors with Combined features

- Some detectors combine features in a **single kernel** to deal with several issues
  - Edge orientation an contrast
  - Gradient orientation and magnitude
  - Noise
  - Scale

Marr-Hildreth Detector

- Combines
  - A smoothing **Gaussian function**
  - A **Laplacian derivative operator**
- The **kernel is larger** than the previous ones
- Standard deviation **σ** of Gaussian controls its width
- The larger the **σ**, the larger is the **smoothing**
- After smoothing, **zero-crossings** are detected
- The value of **σ** can be used to extract edge details at different **scales**

Canny Detector

- Multi-step method, that allows detecting edges and attenuating noise
- Sometimes characterized as the **optimal edge detector**
- Also combines **smoothing Gaussian** with a **Laplacian**, but it uses directional information from the **gradient**
- It detects edges through the **zero-crossings** of the image's **2nd derivative**, smoothed with a 1D Gaussian filter in the direction of the gradient

---

Edge Linking

- Edgel **aggregation** can be done sequentially
- A contour is **followed**. At each pixel a **decision** is taken on how to pursue
- An alternative is to perform **aggregation after edge detection**

Local Contour Following

- Expand the neighborhood to detect an edge
- Neighborhood must be
  - Large enough to close **false gaps**
  - Small enough not to link independent fragments

Image Segmentation

- **Subdivide an image into regions** that are meaningful in the application context
- Group pixels with **similar features**
- Humans can easily segment an image - **Parallel processing** + a priori **scene knowledge**
- Simplest computational approaches process **one pixel** at a time, and do not use neighborhood information
- For image segmentation, it is reasonable to start with **low-level pixel by pixel operations**

> No segmentation method can be successfully applied to every case
>
> There is no perfect segmentation method

Grayscale Thresholding

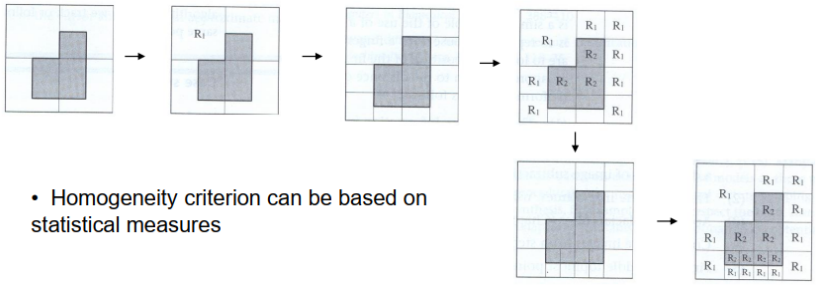
- If the **threshold** value is unknown, analyze the histogram to choose an adequate threshold value
- For a bimodal histogram, the threshold value corresponds to the **valley** between the **peeks**
- This approach can produce **classification errors**, depending on the image histogram and the intensity values of the objects
- The **thresholding** can be applied to image **sub-regions**

Segmentation by Region Growing

- The simplest method for region growing is **pixel aggregation**
- Need for **seed-pixels** and aggregation rules based on
  - Connectivity
  - Similarity of common features
- All pixels that are **neighbors** to the seed pixel and have **similar features**, are **labeled** as belonging to the same region
- The placement of the **initial seed pixels** is an important issue
- For instance, that can be done by the user or as a result of **histogram analysis**

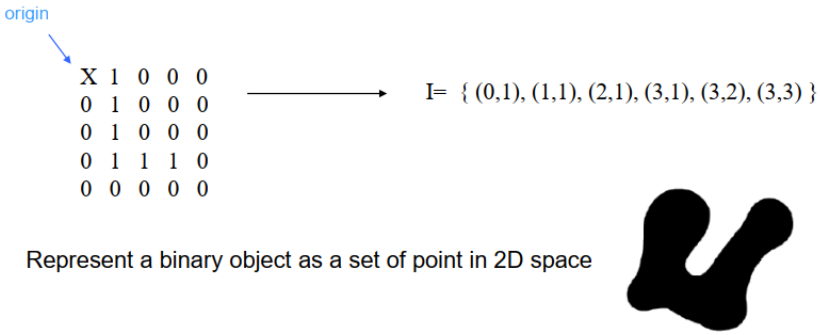
Region Split-and-Merging Segmentation

- Divide and Conquer strategy
- Image **subdivision** until we get **homogeneous regions**
- No need for **seed pixels**
- But **blocky** regions

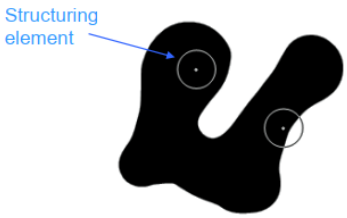


12 - Operações Morfológicas

- Treat images as **sets of points**
- The **modify in a controlled way** the structure / morphology of an image
- They can be easily applied to **binary images**
- Can also be applied to **grayscale** or **color images**
- A binary image can be represented as a set of **2D points** by listing, **row by row**, the **coordinates** of the **foreground**

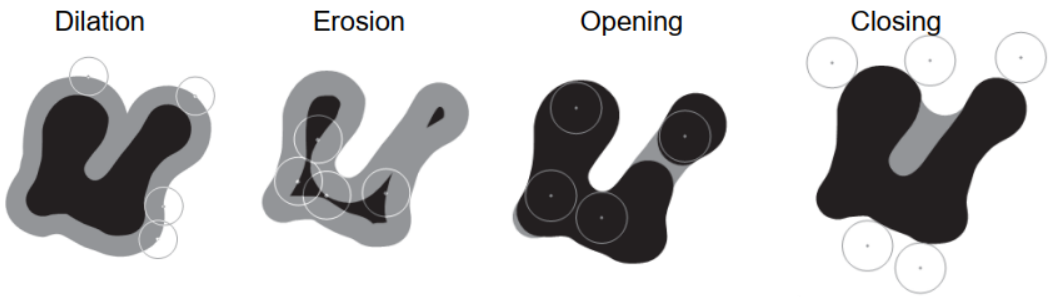


- Some morphological operations can be regarded as **shape filtering**
- The **shape** has to be known a priori
- **Scan** an image with a given **structuring element** (SE)



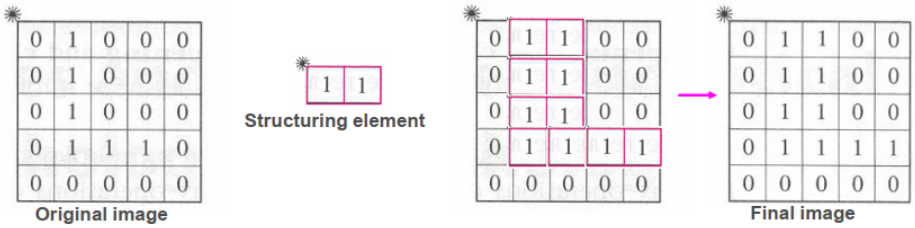
Main Morphological Operations

Basic Operations Composite Operations



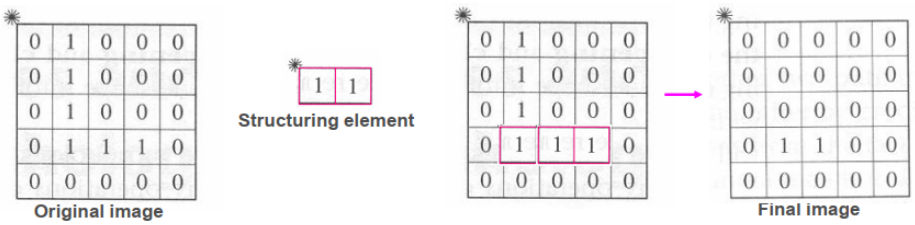
Dilation

- **Replicate the SE** at each original image foreground pixel
- In general, it originates an **expanded** version of the foreground
- Small holes and intrusions are filled
- **-»** Place the origin of the SE at each foreground (1) pixel of image I and copy all SE 1 pixels to the corresponding pixels in the result image



Erosion

- Eroding an object using a symmetrical SE is equivalent to dilating the background
- The eroded object is **shrunk**
- Holes are enlarged and small extrusions are removed
- **-»** Place the origin of the SE at each foreground (1) pixel of image I and set to 1 the corresponding pixel in the result image, whenever the SE pattern exists in the original image



Contour Extraction

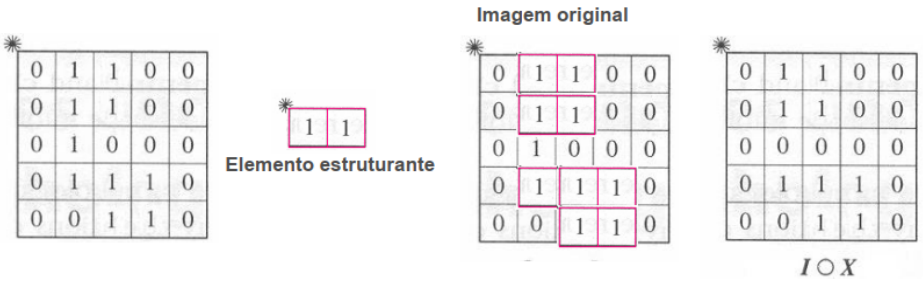
- Since the erosion with an appropriate SE results in a **isotropic shrinking**, foreground contours can be obtained with the **composed operation**

$$I - (I \ominus X)$$

- **Erode** the foreground, then **subtract** it from the original image
- The size of the SE determines the **thickness** of the resulting contours

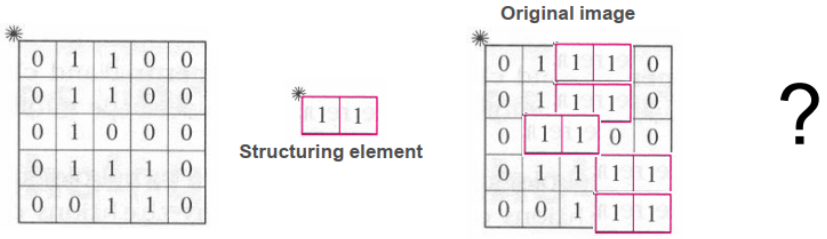
Opening

- Erosion followed by a dilation
- Union of all SEs that **fit inside** the object
- **-»** The SE is applied to the whole object, but no 1 pixel of the SE can appear outside the object



Closing

- Dilation followed by an erosion



Grayscale Morphological Operations

- The same operations
- The **structuring element** represents a **real-valued 2D function**
  - **Positive**, **negative** or **zero** values
  - **Don't care**

Dilation Erosion

$$(I \oplus H)(u,v) = \max_{(i,j)\in H}\{I(u+i,v+j) + H(i,j)\}$$

$$(I \ominus H)(u,v) = \min_{(i,j)\in H}\{I(u+i,v+j) - H(i,j)\}$$