Computer Graphics

A- Shape Representation :-

* Dimension of Objects.

→ Object is set of points in n-dimensional space

→ dim(Object) = k iff ∃ f (One-one) such that $y = f(x)$, $x \in$ Object, $y \in k \neq \dim$ Square

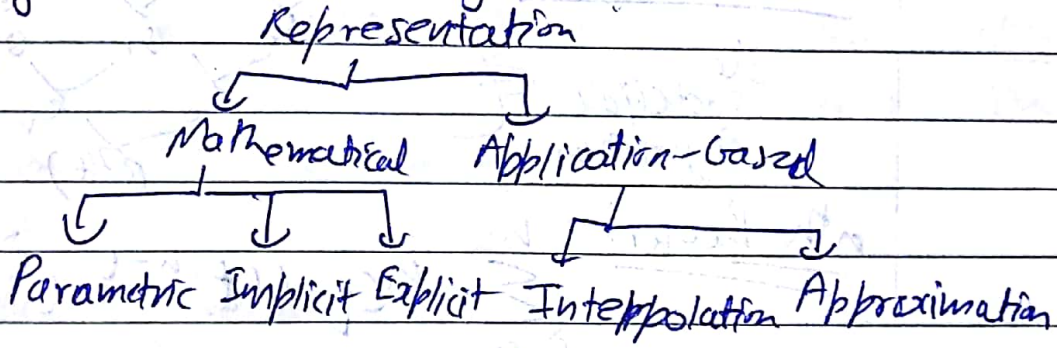
Dim :- 0 1 2 3

Objects :- Point curve surfaces solids.

* Curves

→ Path of continuous moving point in 2D/3D

⇒



Δ Parametric :-

2D → $C(t) = (x(t), y(t))$, $x, y : R \to R$, $C : R \to R^2$

3D → $f(t) = (x(t), y(t), z(t))$, $x, y, z : R \to R$, $f : R \to R^3$.

Eg :- Parametric circle ⇒ $C(t) = (\cos(t), \sin(t))$

⇒ Polynomial :-

If x, y, z are polynomials, C(x, y, z) is poly. param.

$C^n(t)$ is nth deg poly if
$\deg(x) = \deg(y) = \deg(z) = n$

Δ **Implicit:-**

2D ⇒ $C = \{(x,y) \mid c(x,y) = 0\}$,
$c : R^2 \to R$

Example:- $C = \{(x,y) \mid \text{sqrt}(x^2+y^2) - 1 = 0\}$.

Δ **Approximation Curves:-**

↳ Need not interpolate all points.

Eg- Bezier, B-spline, ⟷-Spline

* **Bezier Curve:-**

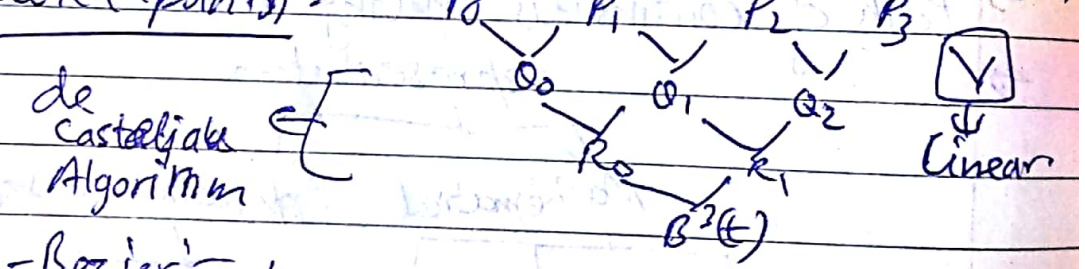
Linear (2 points) :- $B^1(t) = (1-t)P_0 + tP_1$

$P_0 \circ \longrightarrow \circ P_1$

Quadratic (3 points) :- $B^2(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2$

Or, $Q_0 = (1-t)P_0 + tP_1$, $Q_1 = (1-t)P_1 + P_2$, $B^2(t) = Q_0(1-t) + Q_1 t$

Cubic (4 points) :-



de Casteljau Algorithm

nth-Bezier:- $\sum_{i=0}^{n} {}^nC_i (1-t)^{n-i} t^i P_i$

Properties:- * Interpolation of P_0 and P_n.

* Curve straight if all points collinear

* Tangency at first and last section. ~~$\frac{dB}{dt}$~~

$\left.\frac{dP}{dt}\right]_{P_0} = \text{slope}(P_0, P_1)$

$\left.\frac{dP}{dt}\right]_{P_n} = \text{slope}(P_{n-1}, P_n)$

Date:
Page No.:

* Always lies in convex hull.
* Can be split in 2 diff curves anywhere
* $B^n(t)$ can be converted to $B^{n+1}(t)$
* Affine transformation of polygon $\Leftrightarrow$ Bezier curve.

Extra :→ Weighted version.

$$B^n(t) = \frac{\sum_{i=0}^{n} {}^nC_i (1-t)^{n-i} t^i w_i P_i}{\sum_{i=0}^{n} {}^nC_i (1-t)^{n-i} t^i w_i}$$

☺
⇒ Better local control
:- Computation

Δ Interpolation Curves :- Interpolates all points. Eg:- Lagrange, Piecewise Bezier

* Lagrange :- (Only 2D) :-

$$L(x) = \sum_{j=0}^{n} y_j l_j(x).$$

$$l_j(x) = \frac{(x-x_0)}{(x_j-x_0)} \frac{(x-x_1)}{(x_j-x_1)} \cdots \frac{(x-x_{j-1})}{(x_j-x_{j-1})} \frac{(x-x_{j+1})}{(x_j-x_{j+1})} \cdots \frac{(x-x_n)}{(x_j-x_n)}$$

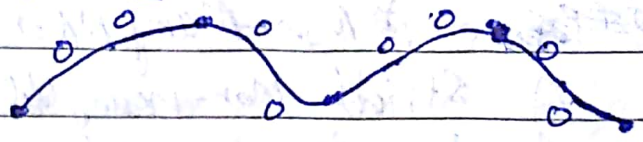
(one term missing)

huge Oscillations and errors.

Piecewise Interpolation :- (poly-curves)

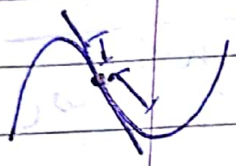
→ Linear :-

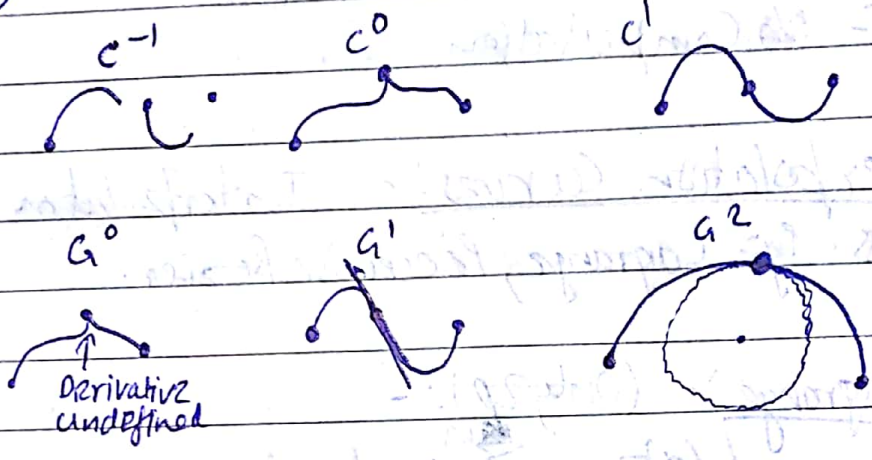
→ Cubic Bezier Poly-curve :-



Continuity in poly-curves:-

- Parametric (C^n):-
  - Tangents ⇒ equal direction & length
  - upto nth derivative defined at interpolation points.
- Geometric (G^n):-
  - Tangents have equal direction.



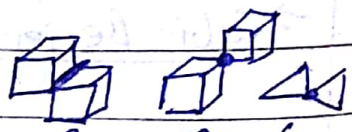


C^n (continuity) ⇒ G^n continuity

G^n continuity ⇒ C^n continuity via reparametrization

* Surfaces:- (Orientable Continuous 2D manifold embed in R^3) OR (Boundary of non-degenerate 3D solid)

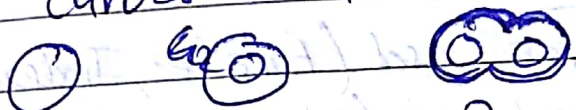
- → Orientable (Diff. Interior & Exterior)
  - ↳ Non-Orientable Eg Mobius Strip.
- → Open / Closed
- → Non-Manifold ⇒



A Strictly non-manifold Point B
B Non-Manifold edge
C Weak non-manifold vertex.

Date:
Page No.:

△ Topological Classification:-
→ Equivalence ⇒ Transformable via stretching and bending.
→ Genus ⇒ Max. cutting along simple curves which don't disconnect

Eg 
Genus 0 1 2

△ Operational Classification:-
↳ Evaluation ⇒ sampling of geometry/attrib.
↳ Modification ⇒ In terms of geometry or topology.
↳ Query → Check point containment.
↳ point distance

△ Parametric ($f: P \to C \mid P \subset R^2, C = f(P) \subset R^3$)
Implicit $f: R^3 \to 0$.

☆ Mesh Representation:- Piecewise linear Approximation, $O(n^2)$

Elements: Faces —Intersect→ Edges —Inte.→ Vertices.
(Subset of 3D planes) (Lines) (Points)

☆ Local Structure:-
↳ Type ⇒ Triangular, Quadilateral, Polygon
↳ shape ⇒ Isotropic (Uniform), Anisotropic.
↳ Density ⇒ Distribution (Uniform / Adaptive)
↳ Alignment and Orientation

☆ Global Structure:-

↳ Topology complexity → (2 manifold / complex non-manifold)
↳ Regularity → (Irregular / semiregular / Highly regular / regular)

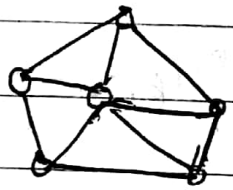
☆ Data Structures:-

↳ Face-based (FaceSet, Indexed Face set)
↳ Edge-based (Winged Edge, Half Edge, Directed Edge)

Concerns → Algorithms, topology data, render/edit, memory req., element access, oriented traversal of edges, incident face edge, vertex access.

⇒ Face set ⇒

Face
$f_1 = (V_1, V_4, V_3)$
$f_2 = (V_1, V_2, V_3)$

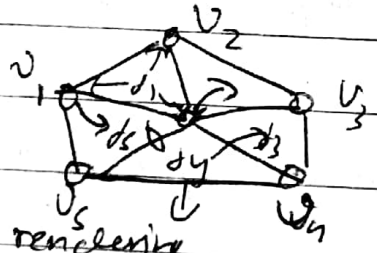


+ Static meshes & rendering
− Duplicate info. No connectivity info

⇒ Indexed Face set:-

Face
Vertices
Faces

Vertex
Point
Face



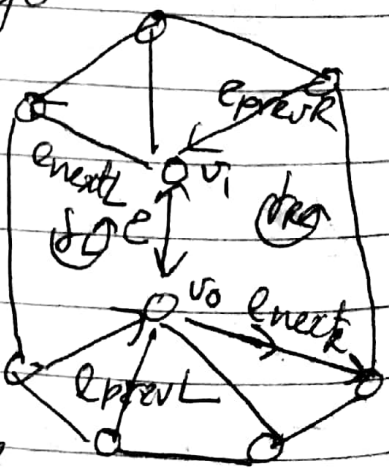
+ Static meshes and rendering
− & No explicit connectivity info.

⇒ Winged Edge.

Vertex
Point
Edge

Face
Edge

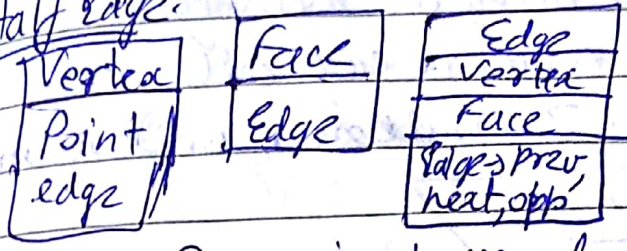
Edge	
V0	V1
fL	fR
eLpred	eRpr
eNext	eNext



→ Arbitrary polygon meshes
→ One-ring traversal, too-many conditions.

Date:
Page No.:

⇒ Half Edge:-



+ One ring traversal, Explicit edge rep.
− Slow rendering

⇒ Directed Edge:-

Half edge → removes references and place in memory so that absolute reference is available.
+ Memory Efficient, one-ring traversal
− Limited usage (Tri/Quad meshes)

Not used

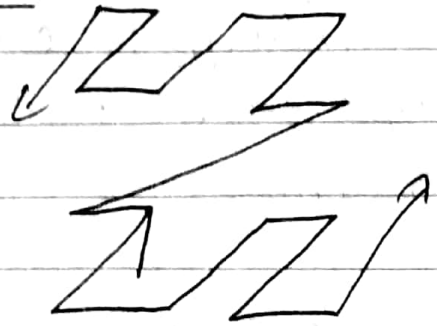
	Rendering	One Ring traversal	Bounding Traversal
→ Face set	Fast	Slow	Slow
→ Indexed face set	Fast	Slow	Slow
* + Topology	Fast	Fast	Slow
M Winged Edge	Med	Slow	Slow
M Quad Edge	Med	Fast	Med
Half Edge	Med/Slow	Fast	Fast
Directed Edge	Med/Slow	Med	Med

* Volumes :- → Spatial Subdivision
→ Implicit (Functional) Rep.
→ Constructive (Hierarchical) Rep.

⇒ Uniform Grids:- 3D N^3 lattice.
→ Find Min, Max coords
→ Define grid resolution.

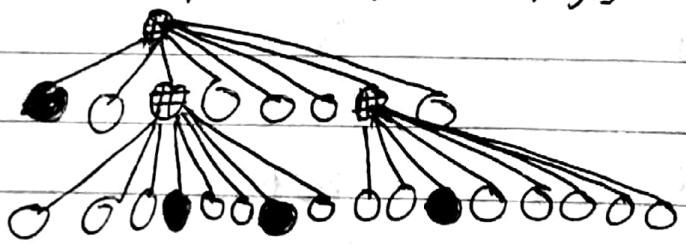
→ Create Array in memory
↳ Store values in each cell

Traversal: Z-index.



+ Trivial OS
+ Parallelizable Algo
+ Bool operations
* – Huge Memory
– Large traversal loops
*

⇒ Octree ⇒ Adaptive hierarchy of cells created within important (non-empty) data regions.



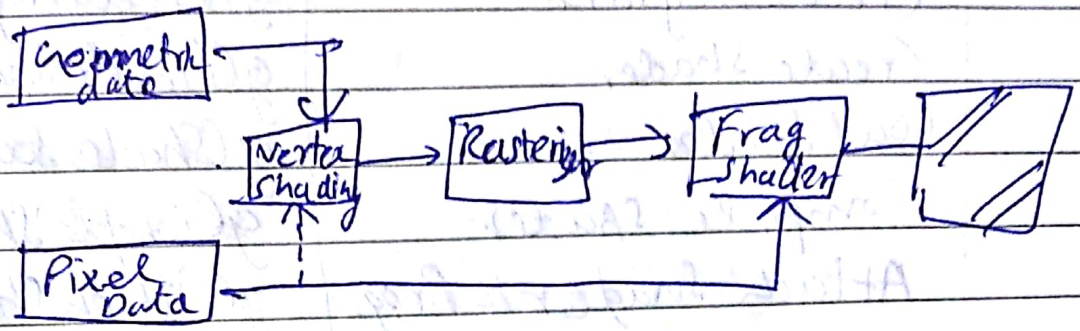
```
Node {
    NodeType type, {Empty, Mixed, Full}
    NodeRef Nodes[8];
}
```

Creation → Top down or Bottom Up
+ Memory efficient
+ Adaptive refinement
– Longer point localization
– Small change may lead to large changes

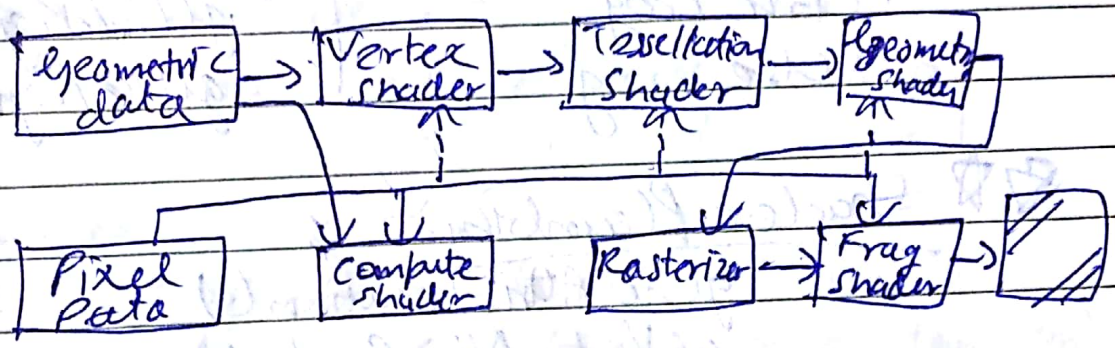
B. # Opengl

OpenGL → Standard (Desktop)
↳ ES (Embedded)
↳ WebGL (JavaScript - Pascal)
↳ SC (Safety Critical Apps)

Pipeline:- (ES/WebGL)



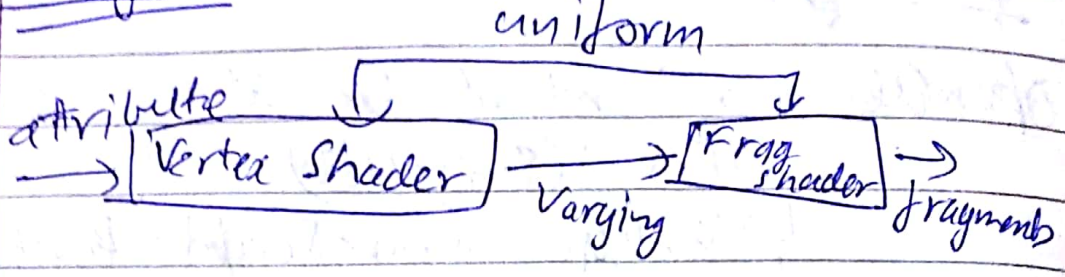
Pipeline:- (std)



→ All APIs are shader based
→ Store data in buffer objects for portability & performance.

Geometric objects:-
represented using vertices.
Vertex → pos-coords
→ Colors
→ Texture coords
↳ Other data.
Position = 4-tuples.
Vertex data → VBO → VAO:

Data flow:-



Creating Shader:-

Create Program	glCreateProgram()
Create Shader	glCreateShader()
Load Shader Source	glShaderSource()
Compile Shader	glCompileShader()
Attach Shader to Prog	glAttachShader()
Link Prog	glLinkProgram()
Use Prog	glUseProgram()

Shader Plumbing:-

glGetAttribLocation()
glVertexAttribPointer().

C. Transformations

Transformation Matrices:-

→ Object Modeling
→ Viewing
→ Projecting

Object Transformations

- Location
- Orientation
- Size
- Translation
- Rotation
- Scaling

Camera Transformation

↳ Location → Translation
↳ Orientation → Rotation

Linear Transformations

$T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $Tp = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax+by \\ cx+dy \end{bmatrix}$

2D Transformations:-

a) Non-Uniform Scaling =>

$$\text{scale}(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

b) Rotation about origin =>

$$\text{rotate}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

c) Translation
↳ not a linear transform.

→ Homogenous coords. (Add extra dimension)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}x + m_{12}y + x_t \\ m_{21}x + m_{22}y + y_t \\ 1 \end{bmatrix}$$

For vectors, make $w = 0$ (To ignore translation)

→ $P_{2D}(x, y) \mapsto P_h(wx, wy, w)$, $w \neq 0$.
Apply homogenised version of 2D transformations to get a new point in w-higher space.

→ For getting 2D point, divide by w'.

Scaling/Rotation ⇒ $\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Translation ⇒ $\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$

Shear ⇒ (skewing).

$Shear_x = \begin{bmatrix} 1 & \tan\phi & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $Shear_y = \begin{bmatrix} 1 & 0 & 0 \\ \tan\phi & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Inverse transformations:-

Scale :- $\begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$, Rotation = $\begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Translation :- $\begin{bmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{bmatrix}$

☆ → Transformation can be composed.

$v' = M_1(M_2 v)$ ⇒ $v' = M_1 M_2 v$.

→ Matrix Multiplication is not commutative.

Transformation about Arbitrary point

Rotation (ϕ) → Bring p to origin ⇒ $R_p(\phi) = T(p_x, p_y) R(\phi) T(-p_x, -p_y)$

Rotate ϕ

Send origin to p

Scaling $\Rightarrow$ $S_p(s_x, s_y) = T(p_x, p_y) R(\phi) T(-p_x, -p_y)$

Decomposition of a transformation:-

Given $M = R_1 S R_2$

Find, R_1, S, R_2 =

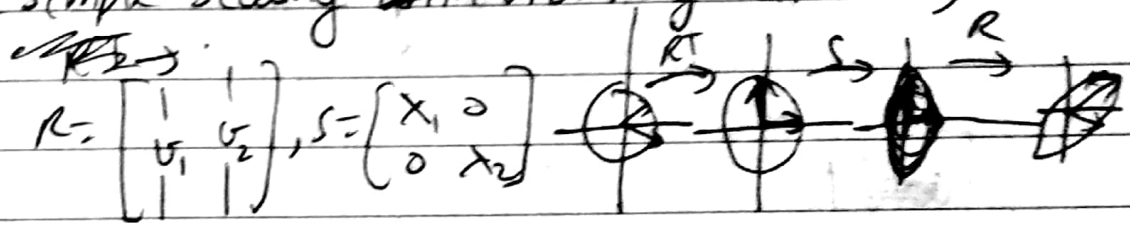
- Find SVD of (M)

Symmetric Eigenvalue Decomposition:-

$A = R S R^T$

$A \Rightarrow$ Symmetric, $R \Rightarrow$ Orthogonal $(RR^T = I)$, S is diagonal

(Simple Scaling in arbitrary direction)

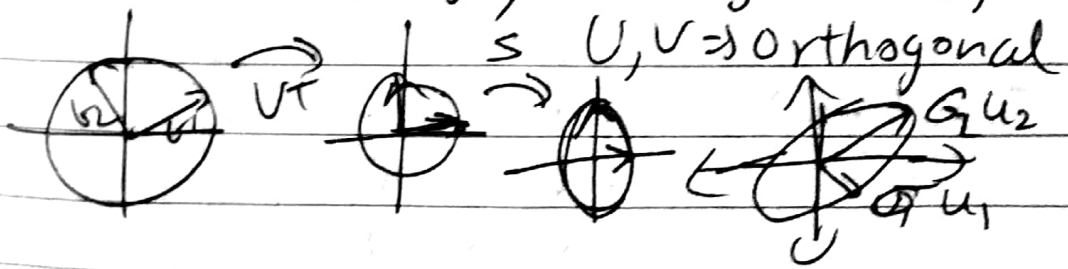


$$\begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \cdot \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

Symmetric Value decomposition:-

$$U = \begin{bmatrix} | & | \\ u_1 & u_2 \\ | & | \end{bmatrix}, V = \begin{bmatrix} | & | \\ v_1 & v_2 \\ | & | \end{bmatrix}, S = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$$

$A = U S V^T$, A: non symmetric, S: diagonal



Polar Decomposition:-

$$\begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} = \begin{bmatrix} 1 & \frac{\cos\phi - 1}{\sin\phi} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin\phi & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{\cos\phi - 1}{\sin\phi} \\ 0 & 1 \end{bmatrix}$$

Image shears through pixel locations

$$\begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} = \begin{bmatrix} i+sj \\ j \end{bmatrix}$$

3D Transformations :-

Scaling :-

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

Rotation :-

$$\begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \text{rotate}_z(\phi)$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \text{rotate}_x(\phi)$$

$$\begin{bmatrix} \cos\phi & 0 & -\sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \text{rotate}_y(\phi)$$

Translation :-

$$\begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

→ Rotation more complex → 3D Sym L·T → SED to RSR^T

→ 3D L·T → SVD to R_1SR_2 → 3D rotation → 3D shears matrices

Date:
Page No.:

Arbitrary 3D Rotations:-

Given $\vec{u}, \vec{v}, \vec{w}$, mutually orthogonal unit vectors.

$$R_{uvw}^{T} = \begin{bmatrix} | & | & | \\ u & v & w \\ | & | & | \end{bmatrix}$$

uvw coord to xyz coord.

Rotation about a vector a:-

→ Rotate uvw → xyz basis.

→ Rotate about z by ϕ

→ Rotate back.

Constructing Basis from single vector

Let $\vec{a}$ be the given vector

→ $w = \frac{a}{|a|}$

→ Take t not collinear with w;

$$u = \frac{t \times w}{|t \times w|}$$

→ $v = w \times u$

Transforming Normals:-

$$N = (M^{-1})^{T}$$

↳ surface transformation

Normal Transformation

D. Viewing:-

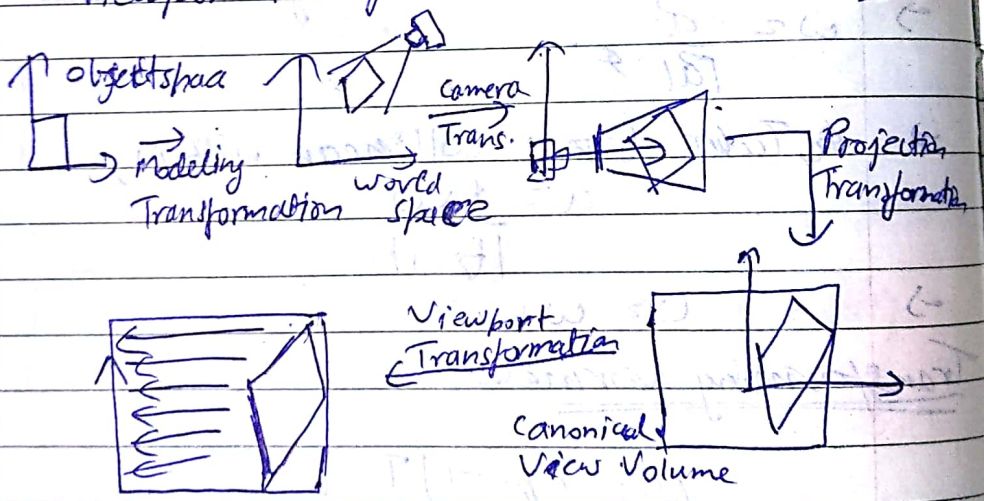
Types of Projection:-

↳ Orthographic [determined by direction of Projection]
↳ Perspective [determined by centre of projection]
(Vanishing Point (View point)
↳ One point
↳ two point
↳ three point

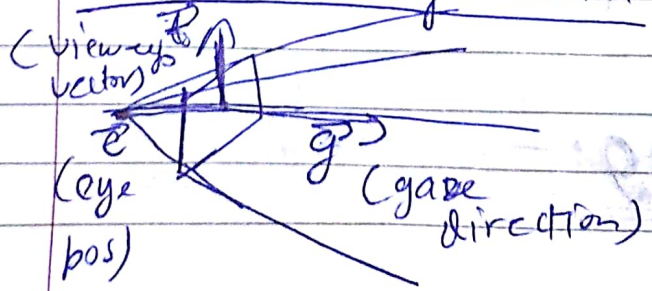
Viewing Transformations:-

Mapping 3D locations in canonical coords to coords in Image.

→ Camera Transformation
→ Aprojection transformation
→ Viewport transformation (or windowing).



⇒ Camera transformation:-



1. Coord system for camera space:-

$$w = \frac{-g}{|g|}, \quad u = \frac{t \times w}{|t \times w|}, \quad v = w \times u$$

uvw → xyz, with origin at e

⇒ $[u \; v \; w \; e]$

$$\begin{matrix} 0 & 0 & 0 & 1 \end{matrix}$$

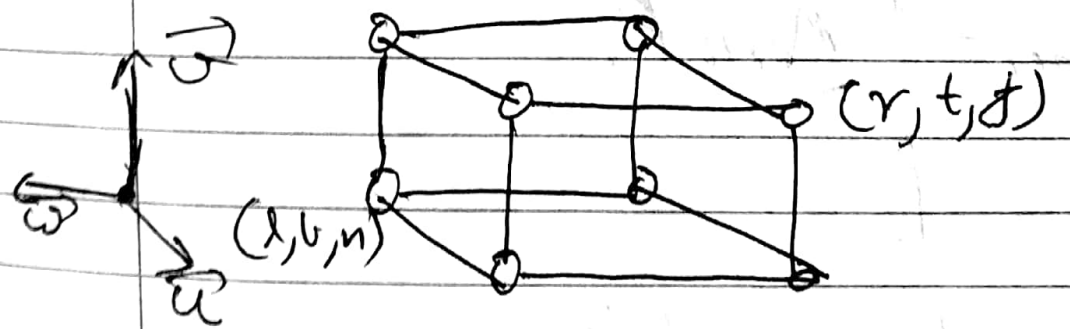
So, xyz to xyzw space :-

$$M_{cam} = [u \; v \; w \; e]^{-1}$$

⇒
$$\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Projection Transformation:-

↳ Orthographic projection:-



$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\left(\frac{r+l}{r-l}\right) \\ 0 & \frac{2}{t-b} & 0 & -\left(\frac{t+b}{t-b}\right) \\ 0 & 0 & \frac{2}{n-f} & -\left(\frac{n+f}{n-f}\right) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

• Size of object :- $y_s = \frac{d}{z} y$

Linear transform :- $x' = ax + by + cz$

Affine transform :- $x' = ax + by + cz + d$

Projective transform / homography :-

$$x' = \frac{ax + by + cz + d}{ex + fy + gz + h}$$

Perspective transform :-

$$P = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ z(n+f) - fn \\ z \end{bmatrix} \sim \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n+f - \frac{fn}{z} \\ 1 \end{bmatrix}$$

$$M_{per} = M_{ortho} P$$

Windowing transformation :-

i) Move to origin (x_t, y_t)

ii) scale by ratio

iii) Move back to (x_t', y_t')

2) $\begin{bmatrix} 1 & 0 & x_l' \\ 0 & 1 & y_l' \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x_h - x_l}{x_h - x_l} & 0 & 0 \\ 0 & \frac{y_h - y_l}{y_h - y_l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_l \\ 0 & 1 & -y_l \\ 0 & 0 & 1 \end{bmatrix}$

Viewport Transformation:-

$$\begin{bmatrix} x_{screen} \\ y_{screen} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y - 1}{2} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{canonical} \\ y_{canonical} \\ 1 \end{bmatrix}$$

$$M_{vp} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How to project:-

$M = M_{vp} M_{per} M_{cam}$

For each line segment (p_i, q_i) do

$p_2 M_{ai}$

$q_2 M_{ai}$

draw line (p_{x_p}, q_{x_q})

E. Rasterization

(Scan conversion)

↳ Enumerate pixels covered

↳ Interpolate attributes to pixels

$O(p) \Rightarrow$ fragments.

Rasterizing Lines:-

Digital Differential Analyzer:-

$$\frac{dy}{dx} = m, \quad m: \text{slope.}$$

$$m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y}{\Delta x}, \quad 0 \le m \le 1$$

So $\Delta y = m \Delta x$. ~~So change~~

Mid-point line algorithm:

Let $f(x, y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0 = 0$

$m \in (0, 1)$,

At a point (x, y), next candidates are $(x+1, y)$ or $(x+1, y+1)$, so, check $f(x+1, y+0.5)$.

→ . $y = y_0$

for $x = x_0$ to x_1 do

draw (x, y)

if $f(x+1, y+0.5) < 0$ then

$y += 1$.

We can see that

$f(x+1, y) = f(x, y) + (y_0 - y_1)$

$f(x+1, y+1) = f(x, y) + (y_0 - y_1) + (x_1 - x_0)$

So,

→ $y = y_0$.

$d = 2f(x_0 + 1, y_0 + 0.5)$

for $x = x_0$ to x_1 do

draw (x, y)

if $d < 0$

update d, $y = y + 1$

⇒ Midpoint circle algorithm:-

$f(x,y) = x^2+y^2-r^2 = 0$.

Draw an octant and replicate.

$x_{i+1} = x_i + 1$

$d_i \geq 0 : y_{i+1} = y_i - 1$

$d_i < 0 : y_{i+1} = y_i$

$d_i = f(x_i+1, y_i - \frac{1}{2})$

Run as before..

⇒ Rasterizing triangles:-

$P_0(x_0,y_0)$, $P_1(x_1,y_1)$, $P_2(x_2,y_2)$.

Color interpolation using Barycentric coordinates $c = \alpha c_0 + \beta c_1 + \gamma c_2$

$\alpha+\beta+\gamma = 1$, $0 \leq \alpha, \beta, \gamma \leq 1$

$\beta = \frac{f_{ac}(x,y)}{f_{ac}(x_b,y_b)}$, $\gamma = \frac{f_{ab}(x,y)}{f_{ab}(x_c,y_c)}$; $\alpha = 1-\beta-\gamma$

Where $f_{ac}(x,y) = (y_a-y_c)x + (x_c-x_a)y + x_a y_c - x_c y_a$

→ Find bounding box:-

for all (x,y) in bounding box,

Calculate α, β, γ for (x,y)

if $(\alpha>0 \wedge \beta>0 \wedge \gamma>0)$.

$c = \alpha c_0 + \beta c_1 + \gamma c_2$

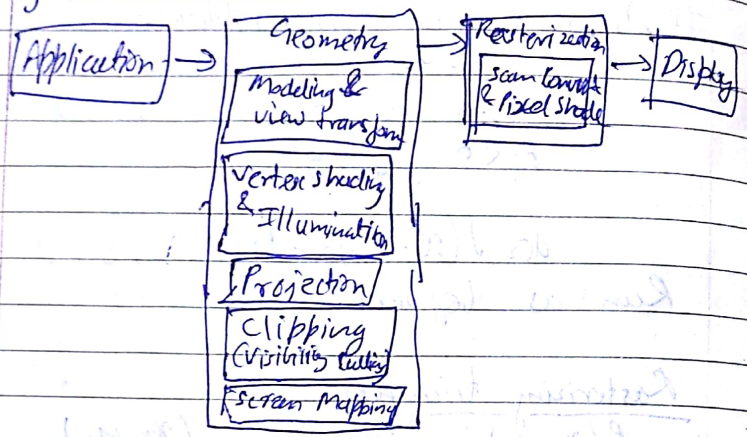
draw

→ For boundary, Draw as line segment

Date:
Page No.:

F. Clipping

Graphics Pipeline:-

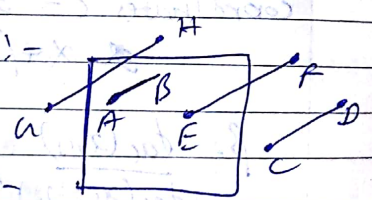


Line segment clipping:-

~~Cases~~

☆ Cohen-Sutherland Clipping:-

← 4 cases →



Create regions as follows:-

1001	1000	1010
0001	0000	0010
0101	0100	0110

$y > y_{max}$

$y < y_{min}$

$x < x_{min}$ $x > x_{max}$

O_1, O_2 ⇒ Outcodes of end points.

⇒ $(O_1 = O_2 = 0)$ ⇒ fully in.

⇒ $(O_1 \neq 0, O_2 \geq 0) \vee (O_1 = 0, O_2 \neq 0)$ ⇒ clip

⇒ $(O_1 \wedge O_2) \neq 0$. Reject.

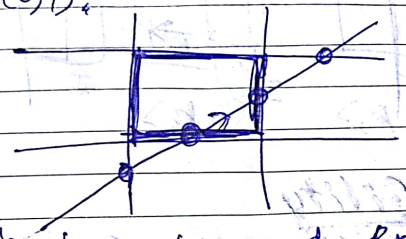
⇒ $(O_1 \wedge O_2) = 0$ More checks

Date:
Page No.:

Liang Barsky Clipping :-

$$p(t) = p_1 + (p_2 - p_1)t$$

$t \in (0, 1)$.



if $t_e < 0$ & $t_l > 1$: Accept
If $t_e > t_l$ or $t_l < 0$: Reject
If $t_e > t_l$: Reject
Else clip.

Find t's for entry and exit

$\Delta x = x_2 - x_1$, $\Delta y = y_2 - y_1$

$L \le x_1 + t\Delta x \le R$, $B \le y_1 + t\Delta y \le T$.

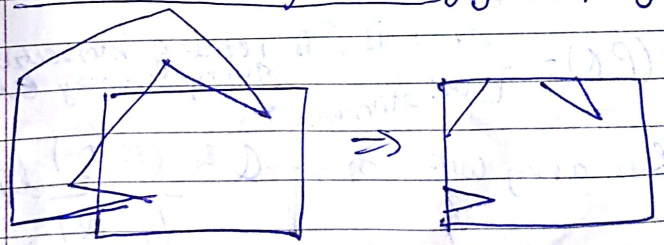
$\downarrow$

$-t\Delta x \le x_1 - L$	$C < 0 \Rightarrow$ out $\rightarrow$ in
$t\Delta x \le R - x_1$	$C > 0 \Rightarrow$ in $\rightarrow$ out
$-t\Delta y \le y_1 - B$	$C = 0 \Rightarrow$ line is //
$t\Delta y \le T - y_1$	

C and Δt ; q

Polygon Clipping :-

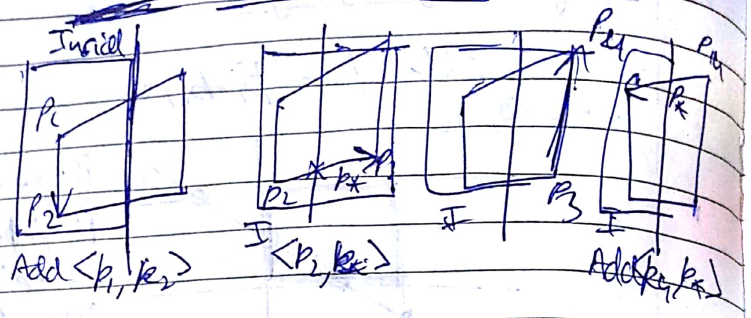
=> Sutherland Hodgman Polygon Clipping.



clip each polygon edge against a clip edge and re-enter algorithm ...

Date:
Page No.:

Clipping cases



G. Visibility

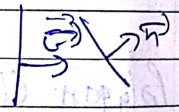
- Hidden Surface Removal
  - Object Precision. OP
  - Image Precision IP
  - List priority. LP

☆ **Back face Culling**

OP → Discard back face polygons from culling.

→ check for direction of $\vec{n}$ vertex w.r.t - view plane.

$\vec{v} \cdot \hat{n} > 0$,



☆ **Ray Casting**

IP

$$V(P,Q) = \begin{cases} 1 : \text{if } P \text{ is result of intersection query on ray} \\ 0 : \text{otherwise} \end{cases}$$

R is a ray with origin Q & $\frac{(P-Q)}{|P-Q|}$ direction.

→ Cast a ray from viewpoint to every pixel and find hitting objects. Store the first hit object.

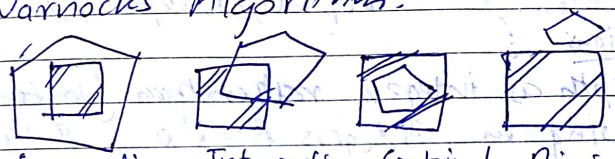
$$r(t) = e + t(p_{ij} - e)$$

Each O_i returns $t_i > 1$ ~~such that first intersection found~~.

☆☆

☆ Warnock's Algorithm.

OP



Surrounding Intersection Contained Disjoint.

→ If all poly disjoint, fill with bg color

→ One intersecting or contained polygon —
  → First fill with bg
  → Scan-convert poly.

→ Only one surrounding polygon, fill area with poly color

→ More than one polygon, then check if one poly overshadows all others, ~~fill~~ draw poly.

→ Else, divide R in four parts and recurse

☆ Z-buffer algorithm

IP

→ Record depth info

→ Z-buffer stores depth.

Framebuffer

Init FB to bg color.
Init DEPTH to ∞
for face F
  for point p of F
    if p projects to FB(i,j)
      if Depth(p) < DEPTH(i,j)
        FB(i,j) = color of F at p
        Depth(i,j) = Depth(p)

Optimizations

depth as integers rather than floats.
using on integers B = {0 ... B-1}.
  → map 0 to z = n
  → map B-1 to z = f
  → z, n, f > 0 wrt camera space.

$\Delta z \frac{(f-n)}{B}$.

Resolution of nearby objects:- smaller bin size:- ① decrease f-n or increase B.

$z = n + f - \frac{fn}{z_w}$, z_w - world depth

so $\Delta z_w \approx \frac{z_w^2 \Delta z}{fn}$

Largest bin size $z_w = f$ $\Delta z_w^{max} \approx \frac{f \Delta z}{n}$
$n \neq 0$

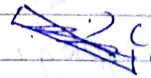
Δz_w^{max}, minimise f and maximise n

(Prob - z-fighting)

Painter's algorithm

LP ⇒ Faces are stored back to front
⇒ Draw the faces from back to front
⇒ Not possible when


Piercing polygons.


Cyclic overlaps

⇒ BSP = tree Algorithm. (Painter's algo implemented)
Binary Space partitioning

- ~~Consider fig~~ Construct Binary tree :—
→ T_1 as root
→ +ve branch ⇒ $f_1(p) > 0$
→ -ve branch → $f_1(p) < 0$.

```
draw (tree, e)
  if tree is empty
      return
  if f_tree.root (e) < 0
      draw (~~tree.left~~) +
      rasterize . tree. root
      draw —
  else
      draw —
      rasterize tree. root
      draw +
```

+ One tree
+ Works for all viewpoints
# Requires splitting. if planes cross.