Computação Gráfica

Computer Graphics

Engenharia Informática (14350) – 3º ano, 1º semestre

Chap. 2 – Geometry Basics

These transparencies were inspired on those due to Machiraju, Zhang, and Möller at:
https://www.cs.sfu.ca/~torsten/Teaching/Cmpt361/LectureNotes/PDF/05_geom_basics.pdf

Overview

. . .:

- Scalar, point, and vector
- Vector space and affine space
- Basic point and vector operations
- Lines, planes, and triangles
- More generic geometric objects
- Rasterization
- Modern OpenGL pipeline

Scalar, point, and vector

Scalar:

- Definition: a quantity, e.g., edge length

Point:

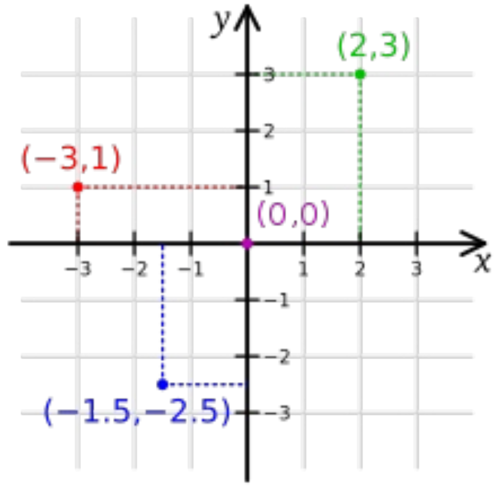
- Definition: a location in space
- specified by an k-tuple
- always given with respect to some coordinate system

Vector:

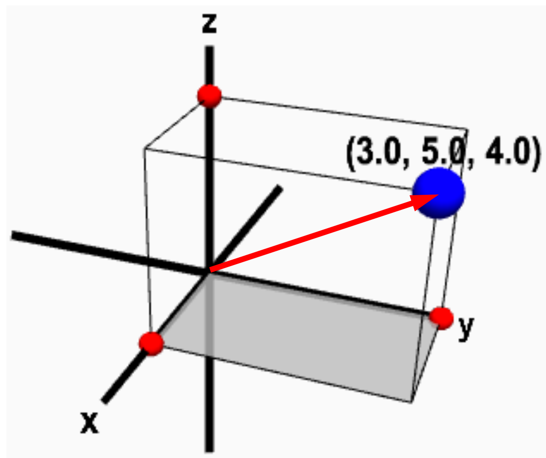
- Definition: a directed line segment between points

Spaces:

- Examples: vector space, affine space, Euclidean space, etc.



$$P = \begin{pmatrix} x \\ y \end{pmatrix}$$



$$Q = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Vector space

Definition:

- A set of vectors with scalar multiplications and vector additions

Operations:

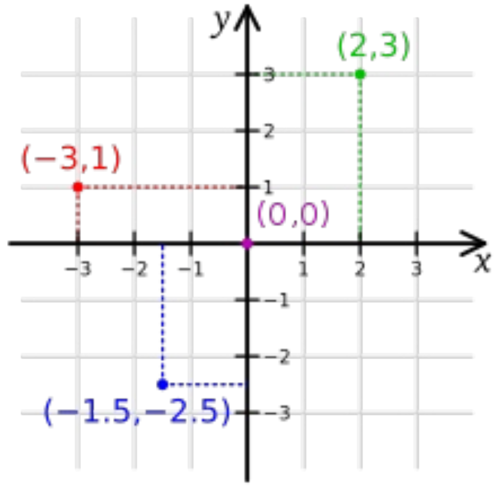
- Scalar-vector multiplication: $\boldsymbol{u} = \alpha\boldsymbol{v}$
- Vector-vector addition: $\boldsymbol{w} = \boldsymbol{u} + \boldsymbol{v}$

Composition:

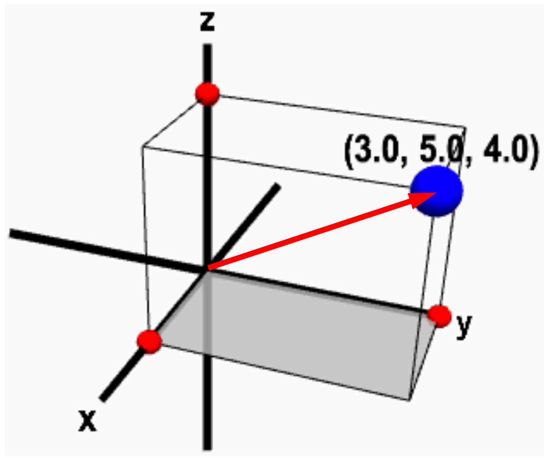
- Expressions like $\boldsymbol{t} = \boldsymbol{u} + 2\boldsymbol{v} - \boldsymbol{w}$ make sense in a vector space

Issue:

- But vectors lack position.
- Inadequate for representing geometry – we need positions, which are given by points



$$P = \begin{pmatrix} x \\ y \end{pmatrix}$$



$$Q = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

We need points to represent geometric objects

Affine space

Definition:

- a vector space + points.

Operations:

- Scalar-vector multiplication
- Vector-vector addition
- *Point-vector addition*
- *Affine sum of points* and *convex sums*

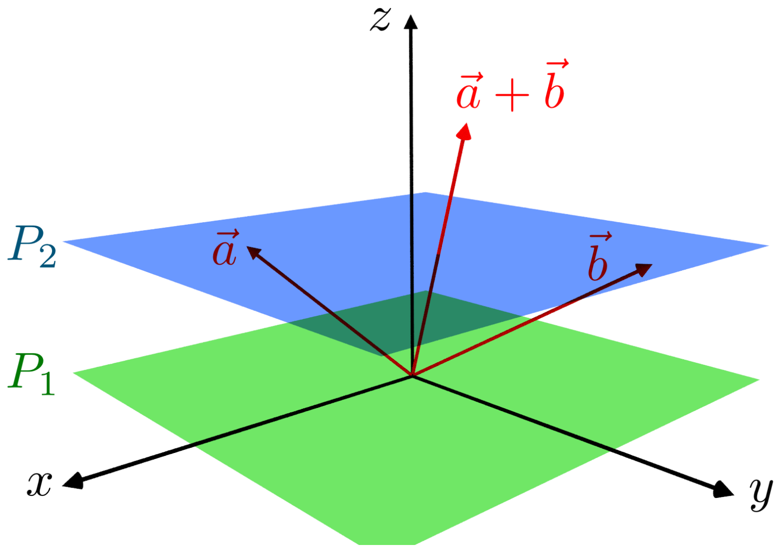
Invariant:

- paralelism

Special affine space:

- Euclidean space: a vector space + distance/norm

https://en.wikipedia.org/wiki/Affine_space



While the lower (green) plane P_1 is a vector subspace of $\mathbf{R}^3$, this is not true for the upper (blue) plane P_2: For any two vectors $a,b \in P_2$ we find $a+b \notin P_2$. However, P_2 is an example of an affine space. The difference $a-b$ of two of its elements lies in P_1 and constitutes a displacement vector.

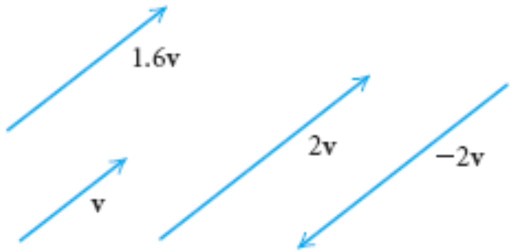
Basic point and vector operations

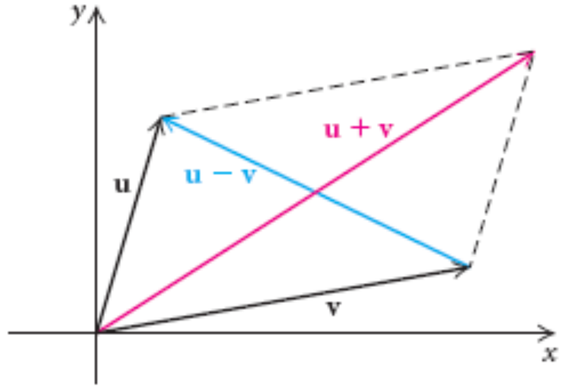
Mixing operations:

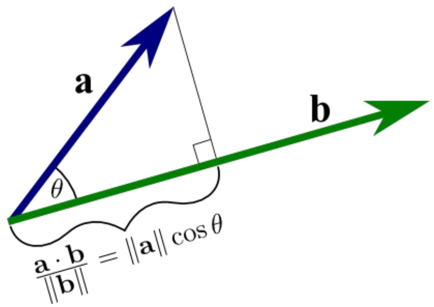
- point – point = vector
- point + vector = point

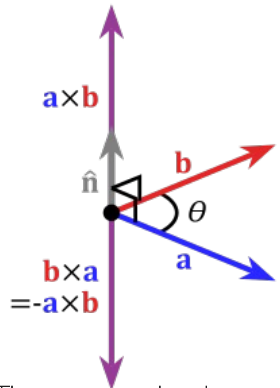
Vector operations:

- scalar * vector = vector
- vector + vector = vector
- vector · vector = scalar (dot product)
- vector x vector = vector (cross product)









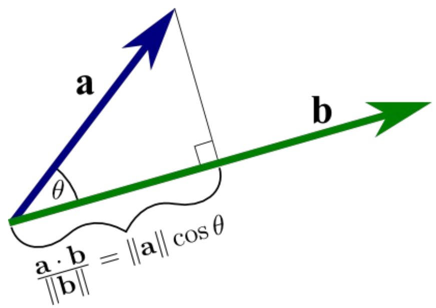
The cross product in respect to a right-handed coordinate system

https://mathinsight.org/dot_product
https://en.wikipedia.org/wiki/Cross_product
https://www.math10.com/en/geometry/vectors-operations/vectors-operations.html

More on dot product

Expressions:

- $\boldsymbol{a} \cdot \boldsymbol{b} = \|\boldsymbol{a}\|\|\boldsymbol{b}\| \cos \boldsymbol{\theta}$
- $\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{a}_x \boldsymbol{b}_x + \boldsymbol{a}_y \boldsymbol{b}_y + \boldsymbol{a}_z \boldsymbol{b}_z$

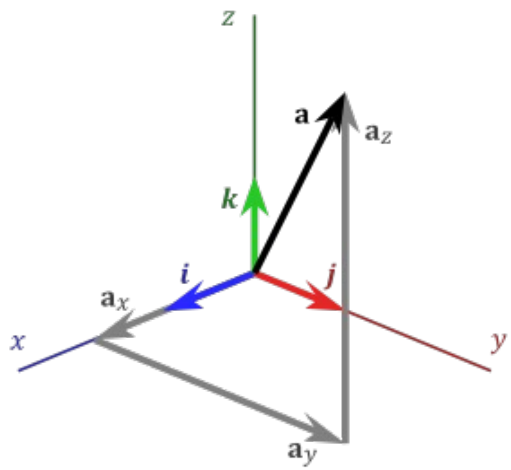


Geometric meanings:

- Two vectors are orthogonal iff $\boldsymbol{a} \cdot \boldsymbol{b} = 0$
- If $\boldsymbol{b}$ is normalized ($\|\boldsymbol{b}\| = 1$), then $\boldsymbol{a} \cdot \boldsymbol{b}$ yields the projection of $\boldsymbol{a}$ in the direction of $\boldsymbol{b}$
- $\boldsymbol{a} \cdot \boldsymbol{a} = \|\boldsymbol{a}\|^2$ is always non-negative

More on cross product

https://en.wikipedia.org/wiki/Cross_product

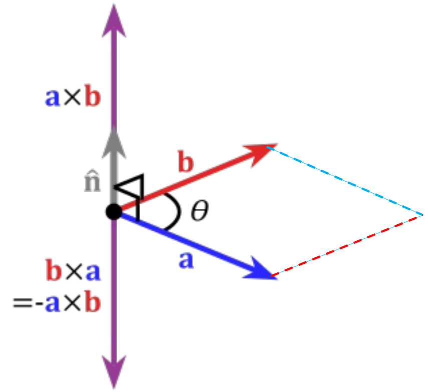


Expressions:

- $\boldsymbol{a} \times \boldsymbol{b} = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ \boldsymbol{a_1} & \boldsymbol{a_2} & \boldsymbol{a_3} \\ \boldsymbol{b_1} & \boldsymbol{b_2} & \boldsymbol{b_3} \end{vmatrix}$
- $\|\boldsymbol{a} \times \boldsymbol{b}\| = \|\boldsymbol{a}\| \|\boldsymbol{b}\| \sin \boldsymbol{\theta} =$ area of the parallelogram
- $\boldsymbol{a} \times \boldsymbol{b} = -\boldsymbol{b} \times \boldsymbol{a}$

Geometric meanings:

- The direction of the cross product is determined by the *right hand rule*
- Cross product $\boldsymbol{a} \times \boldsymbol{b}$ is a vector perpendicular to $\boldsymbol{a}$ and $\boldsymbol{b}$ – frequently used to compute the *normal to a plane*



Affine and convex sums

Affine sum:

- Addition of two arbitrary points is not defined in an affine space
- However, given two points $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$, we can always find a point $\boldsymbol{P}$ between $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$ as follows:

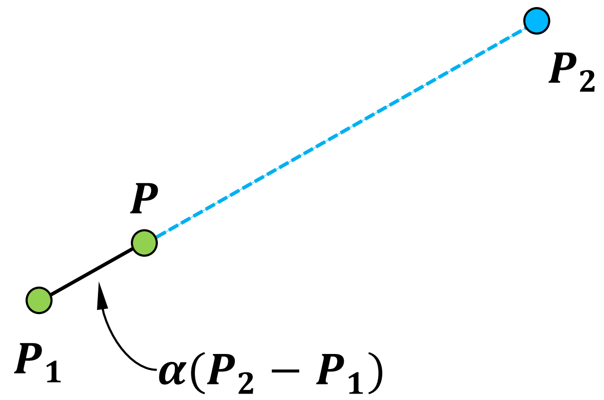
$$\boldsymbol{P = P_1 + \alpha(P_2 - P_1)} \quad \text{or} \quad \boldsymbol{P = (1-\alpha)P_1 + \alpha P_2}$$

  with $\boldsymbol{\alpha} \in \mathbb{R}$

- Thus, *affine sum* (combination) of points can be defined as:

$$\boldsymbol{\alpha_1 P_1 + \alpha_2 P_2 + \cdots + \alpha_n P_n}$$

  with $\boldsymbol{\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1}$



$$\boldsymbol{0 \leq \alpha \leq 1}$$

Convex sum:

- Thus, *convex sum* (combination) of points can be defined as:

$$\boldsymbol{\alpha_1 P_1 + \alpha_2 P_2 + \cdots + \alpha_n P_n}$$

  with $\boldsymbol{\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1}$ and $\boldsymbol{\alpha_i \geq 0}$ for all $\boldsymbol{i}$

Triangle and barycentric coordinates

Convex sum of 3 points:

- However, given three points $\boldsymbol{P_1}$, $\boldsymbol{P_2}$, and $\boldsymbol{P_3}$, we can always find a point $\boldsymbol{P}$ in the triangle $\Delta \boldsymbol{P_1 P_2 P_3}$ as follows:

$$\boldsymbol{P = \alpha_1 P_1 + \alpha_2 P_2 + \alpha_3 P_3}$$

with $\boldsymbol{\alpha_1 + \alpha_2 + \alpha_3 = 1}$

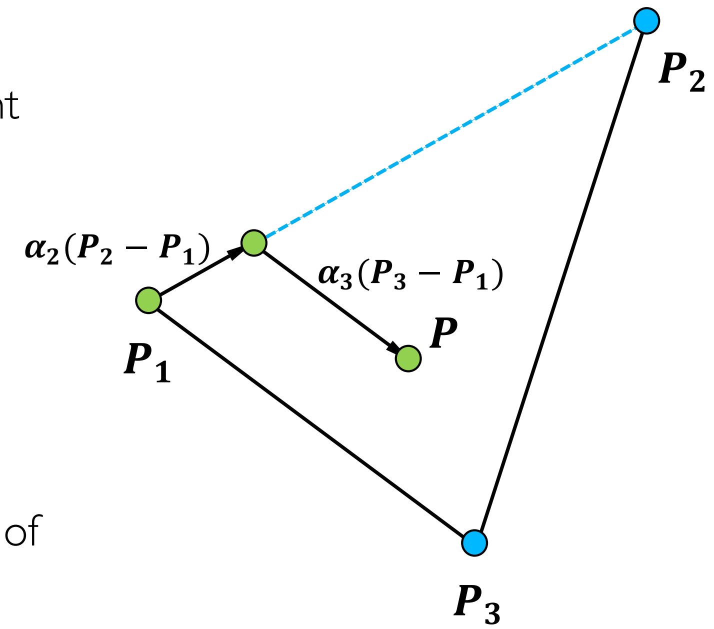
- According to the definition of affine sum (combination), this point $\boldsymbol{P}$ can be defined as follows:

$$\boldsymbol{P = P_1 + \alpha_2 (P_2 - P_1) + \alpha_3 (P_3 - P_1)}$$

- Example: on right-hand side, we and have the point $\boldsymbol{P}$ generated when

$$\boldsymbol{\alpha_1 = \alpha_2 = \frac{1}{4}} \quad \text{and} \quad \boldsymbol{\alpha_3 = \frac{1}{2}}$$

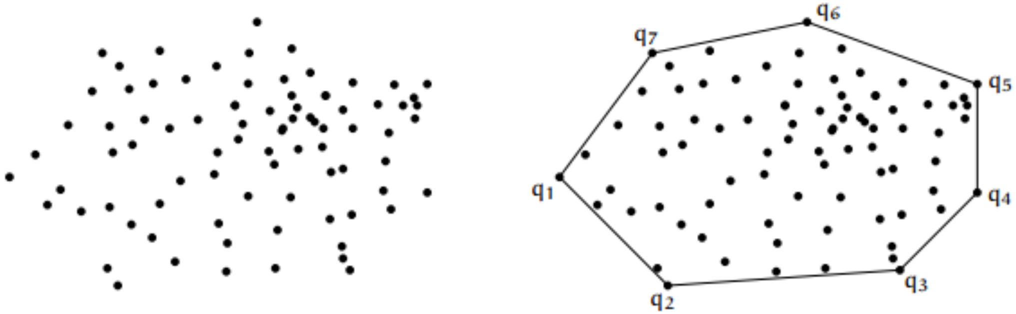
- The weights $\boldsymbol{\alpha_1, \alpha_2,}$ and $\boldsymbol{\alpha_3}$ are called barycentric coordinates of the triangle $\Delta \boldsymbol{P_1 P_2 P_3}$



http://graphics.idav.ucdavis.edu/education/GraphicsNotes/Affine-Combinations/Affine-Combinations.html

Convex hull

https://www.ti.inf.ethz.ch/ew/Lehre/CG13/lecture/Chapter%203.pdf



(a) Input. (b) Output.

Definition:

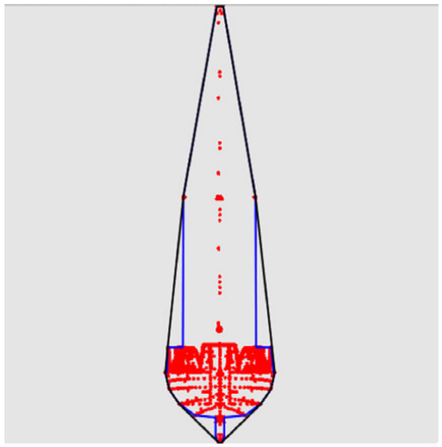
- *Convex hull* of a set of points: set of convex combination of these points

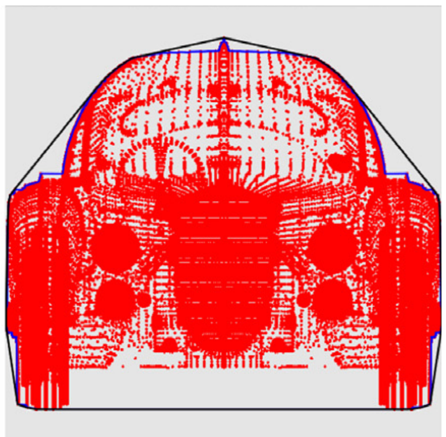
Another definition:

- Alternatively, the *convex hull* is the smallest convex object containing the set of points

Applications:

- Fast collision detection in digital games and robotics.





https://www.sciencedirect.com/science/article/pii/S001044851500113X

Straight line representations

Given two points $\boldsymbol{P}$ and $\boldsymbol{Q}$ on the line , we have:

Parametric: (affine sum)

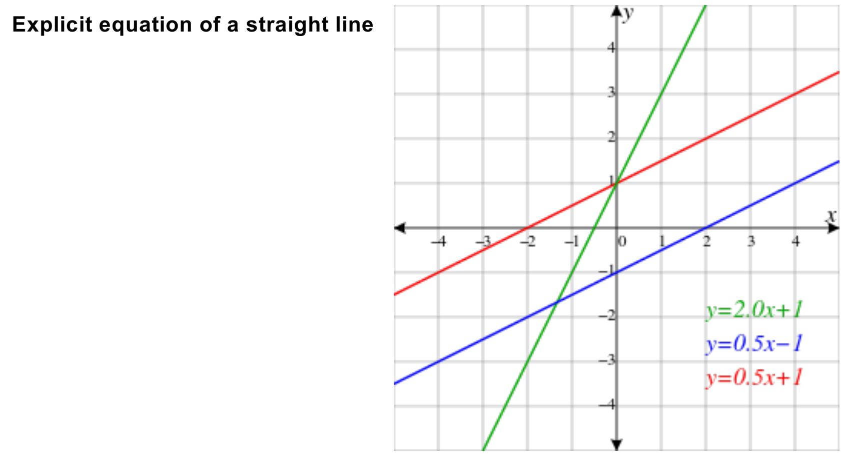
$$\boldsymbol{p}(t) = \boldsymbol{P} + t(\boldsymbol{Q} - \boldsymbol{P})$$

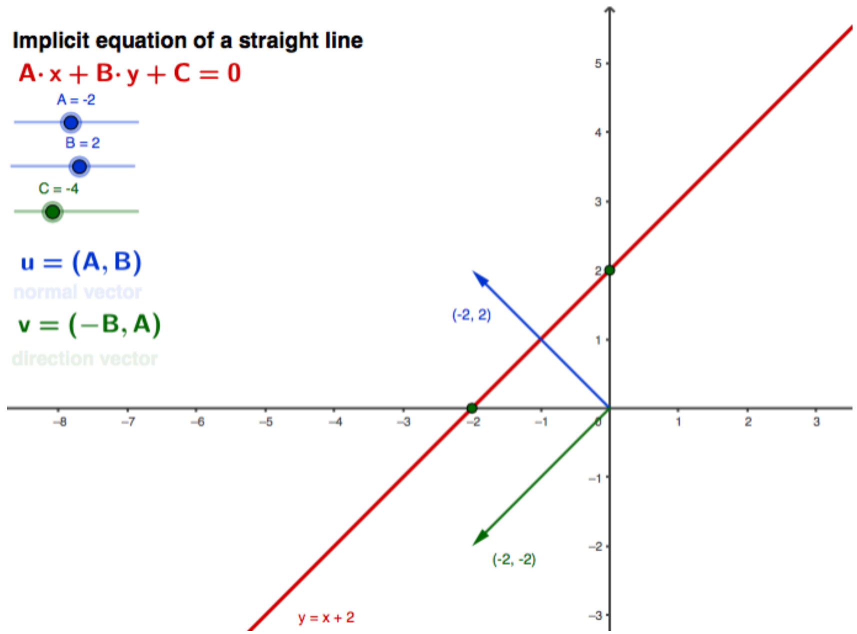
Explicit:

$$y = mx + b$$

Implicit:

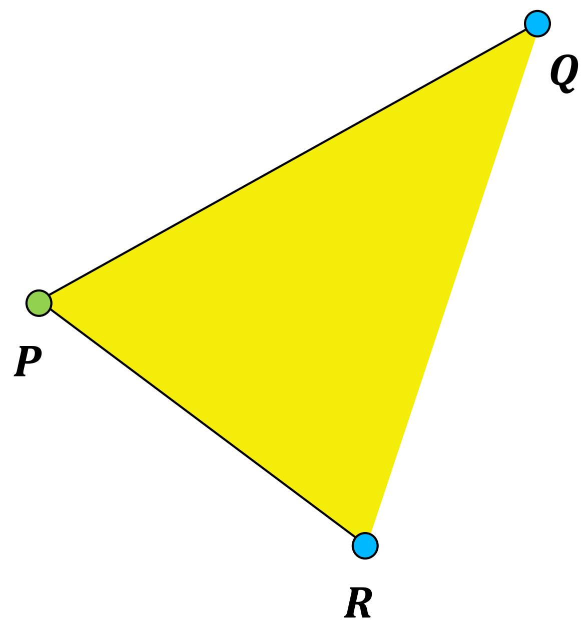
$$Ax + By + C = 0$$





https://en.wikipedia.org/wiki/Line_(geometry)
https://www.geogebra.org/m/y9wjypyd

Plane representations



Given three points ***P,Q,*** and ***R*** on the plane, we have:

Parametric: (affine sum)

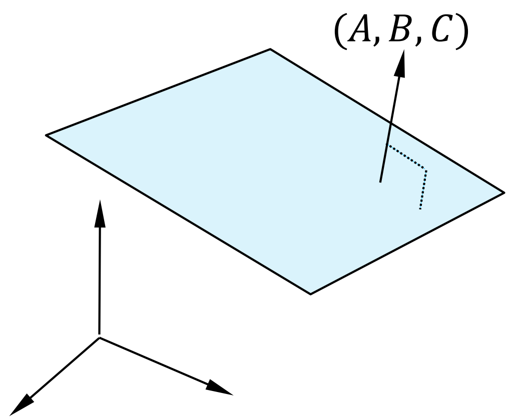
$$\boldsymbol{p}(t,u) = \boldsymbol{P} + t(\boldsymbol{Q} - \boldsymbol{P}) + u(\boldsymbol{R} - \boldsymbol{P})$$

Implicit:

$$f(x,y,z) = Ax + By + Cz + D = 0$$

- Typically, (A,B,C) is the normal vector of the plane
- If $f(x_0,y_0,z_0) \geq 0$, the point (x_0,y_0,z_0) is above the plane
- If $f(x_0,y_0,z_0) \leq 0$, the point (x_0,y_0,z_0) is below the plane
- The distance from (x_0,y_0,z_0) to the plane is given by

$$d = \frac{||Ax_0 + By_0 + Cz_0 + D||}{\sqrt{A^2 + B^2 + C^2}}$$



https://en.wikipedia.org/wiki/Line_(geometry)
https://www.geogebra.org/m/y9wjypyd

Implicit surfaces

Definition:

- An implicit surface is a zero set of a function: $f(x,y,z)=0$
- Example: the unit sphere $f(x,y,z)=x^2+y^2+z^2-r^2=0$

Other designations:

- Isosurface / Level set

Unit surface normal:

- It is the normalized gradient vector

$$\boldsymbol{n} = \frac{\nabla f}{\|\nabla f\|}, \text{ where } \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{bmatrix}$$

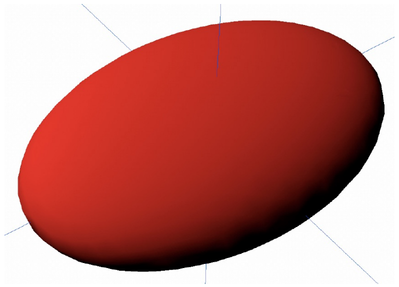
Advantages:

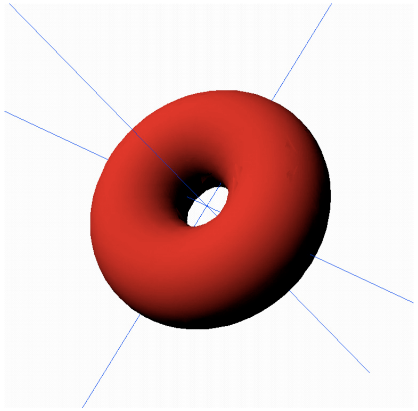
- The entire surface is represented by a single function.
- We can perform interesting operations with this function.
- Example: adding multiple surface functions together where

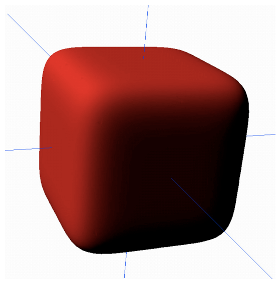
Implicit as solids

Representation of solids:

- Implicit functions represent important classes of solids.
- Example:
  - Ellipsoid: is a closed, manifold surface that encloses a solid.
- The surface of such a solid is said to be its boundary, which separates the interior from the exterior of the solid.







Quadric surfaces

They are a particular case of implicit surfaces.

http://en.wikipedia.org/wiki/Quadric

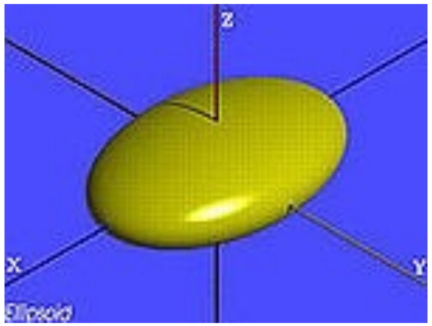
Definition:

- Every quadric surface is defined by the 2nd degree polynomial:

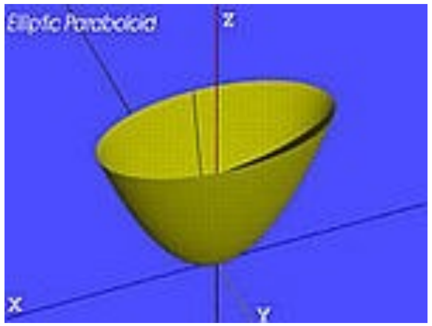
$$f(x,y,z) = Ax^2 + 2Bxy + 2Cxz + 2Dx + Ey^2 + 2Fyz + 2Gy + Hz^2 + 2Iz + J = 0$$

Matrix form:

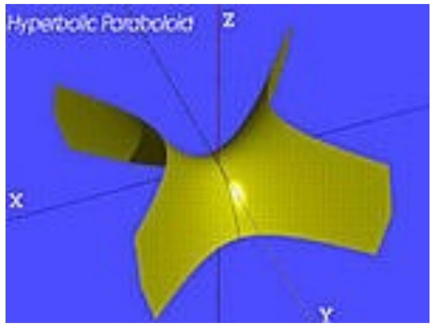
$$f(x,y,z) = v^T M v = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} A & B & C & D \\ B & E & F & G \\ C & F & H & I \\ D & G & I & J \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$



$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - z = 0$$



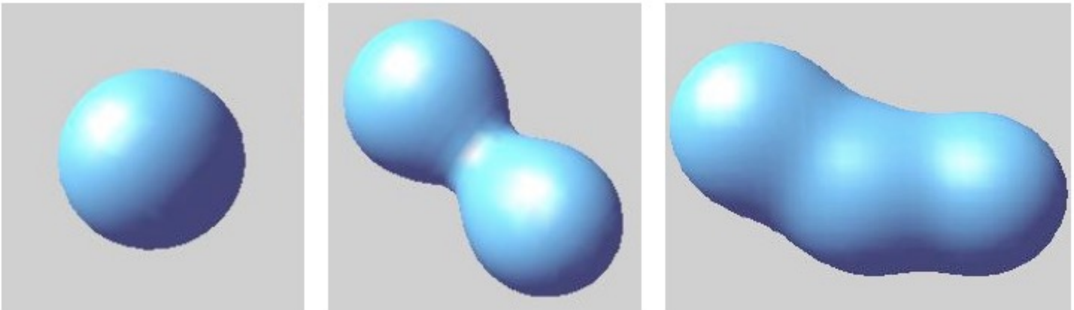
$$\frac{x^2}{a^2} - \frac{y^2}{b^2} - z = 0$$

Gaussian blob surfaces

They are another particular case of implicit surfaces.

Definition:

- A Gaussian blob surface is defined by summing up Gaussian functions for a given threshold T, each one of which is associated to a point (e.g., center of an atom) in 3D space
- $f(x,y,z) = \sum_{i=1}^{N} f_i - T = 0$ with $f_i(x,y,z) = ae^{-\left(\frac{(x-x_0)^2}{2\sigma_x^2}+\frac{(y-y_0)^2}{2\sigma_y^2}+\frac{(z-z_0)^2}{2\sigma_z^2}\right)}$



$N=1$ $N=2$ $N=3$

http://en.wikipedia.org/wiki/Gaussian_function

How is the geometry really rendered on screen?

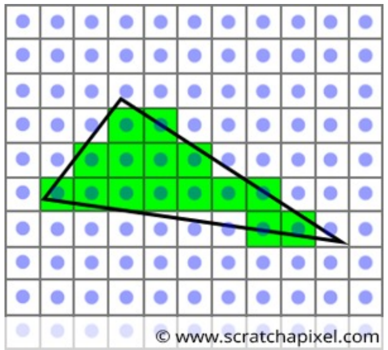
Raster display

Definition:

- Discrete grid of elements (frame buffer of pixels).
  - Shapes drawn by setting the "right" elements
  - Frame buffer is scanned, one line at a time, to refresh the image (as opposed to vector display)

Properties:

- Difficult to draw smooth lines
- Displays only a discrete approximation of any shape
- Refresh of entire frame buffer



Raster terminology

Pixel: Picture Element

- Smallest accessible element in picture.
- Usually rectangular or circular.

Aspect Ratio:

- Ratio between physical dimensions of pixel (not necessarily 1).

Dynamic Range:

- Ratio between minimal (not zero!) and maximal light intensity emitted by displayed pixel

Resolution:

- Number of distinguishable rows and columns on a device measured in:
  - Absolute values (nxm)
  - Relative values (e.g., 300 dpi)
- Usually rectangular or circular.

Screen space:

- Discrete 2D Cartesian coordinate system of screen pixels.

Object space:

- Discrete 3D Cartesian coordinate system of the domain or scene or the objects live in.

Rasterization (or scan conversion)

Definition:

- The process of converting geometry into pixels.
- From screen coordinates (float) to pixels (int)
- Writing pixels into frame buffer.

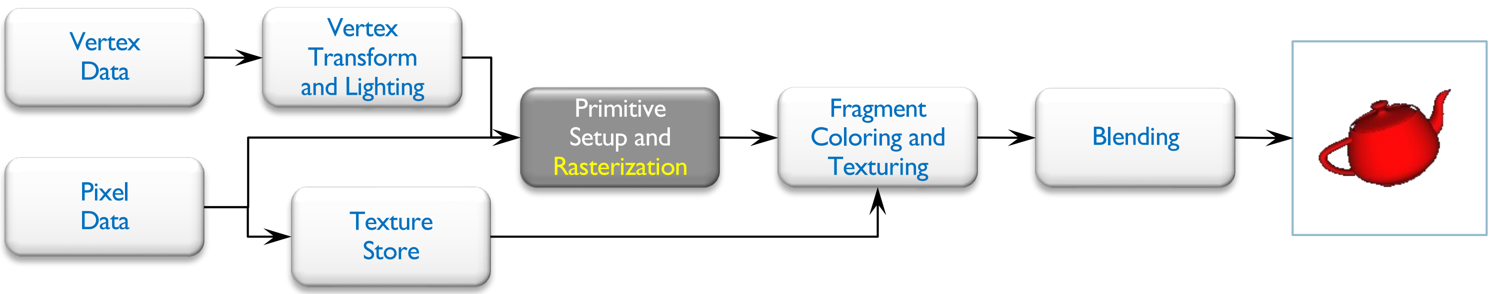
Rasterization:

- Figuring out which pixels to turn on.

Shading:

- Determine a color for each filled pixel.

Modern OpenGL Pipeline:



Graphics primitives

OpenGL Primitive Taxonomy:

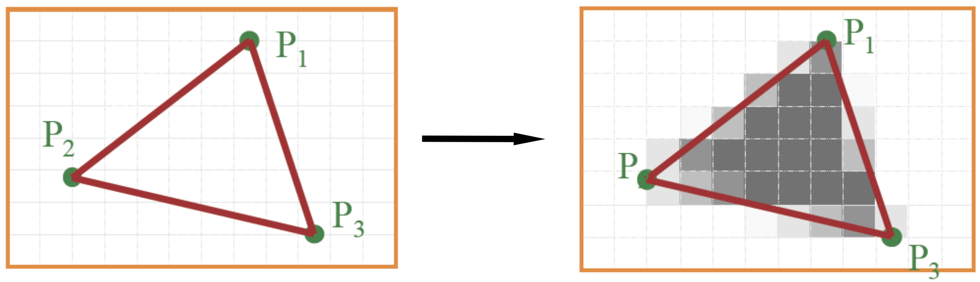
- Point: `POINTS`
- Line: `LINES`, `LINE_STRIP`, `LINE_LOOP`
- Triangle: `TRIANGLES`, `TRIANGLE_STRIP`, `TRIANGLE_FAN`
- Polygon: `QUADS`, `QUAD_STRIP`, `POLYGON`

Other Primitives:

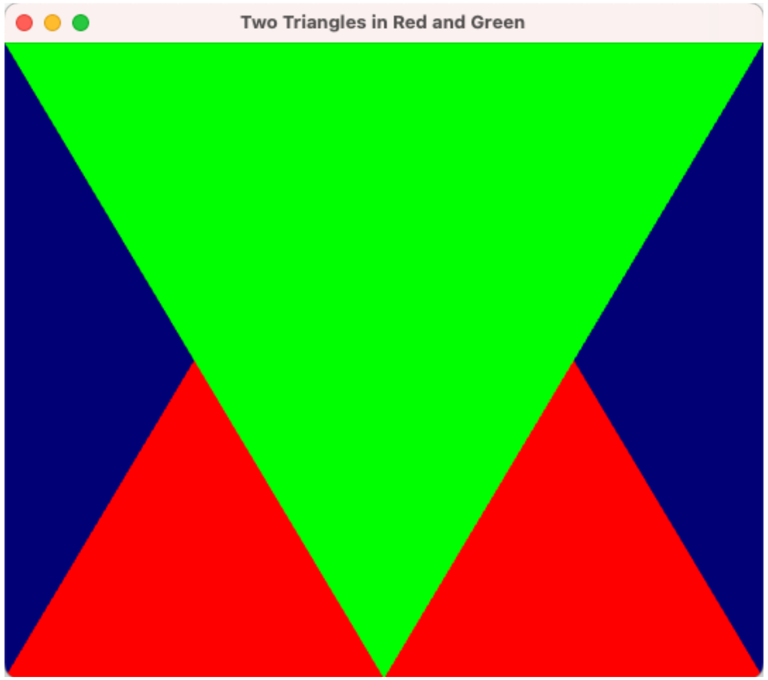
- Any other geometric object must be discretized somehow into points, lines, triangles, quadrangles, and polygons.

Most used primitive:

- triangle



Example in OpenGL



P01, Exercise 1:
Graphics application to draw 2 triangles w/ different colors

```
// Include standard headers
#include <stdio.h>
#include <stdlib.h>

// Include GLEW
#include <GL/glew.h>

// Include GLFW
#include <GLFW/glfw3.h>
GLFWwindow* window;

// GLM header file
#include <glm/glm.hpp>
using namespace glm;

// shaders header file
#include <common/shader.hpp>

// Vertex array object (VAO)
GLuint VertexArrayID;

// Vertex buffer object (VBO)
GLuint vertexbuffer;

// color buffer object (CBO)
GLuint colorbuffer;

// GLSL program from the shaders
GLuint programID;
```

```
// function prototypes
void transferDataToGPUMemory(void);
void cleanupDataFromGPU();
void draw();
```

P01, Exercise 1: Graphics application to draw 2 triangles w/ different colors

```
int main( void )
{
        // Initialise GLFW
        glfwInit();
        // Setting up OpenGL version and the like
        glfwWindowHint(GLFW_SAMPLES, 4);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
        glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE); // To make MacOS happy; should not be needed
        glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
        // Open a window
        window = glfwCreateWindow( 1024, 768, "Two Triangles in Red and Green", NULL, NULL);
        // Create window context
        glfwMakeContextCurrent(window);
        // Initialize GLEW
        glewExperimental = true; // Needed for core profile
        glewInit();
        // Ensure we can capture the escape key being pressed below
        glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);
        // Dark blue background
        glClearColor(0.0f, 0.0f, 0.4f, 0.0f);
        // transfer my data (vertices, colors, and shaders) to GPU side
        transferDataToGPUMemory();
        // render scene for each frame
        do{     // drawing callback
                draw();
                // Swap buffers
                glfwSwapBuffers(window);
                // looking for input events
                glfwPollEvents();
        } while (glfwGetKey(window, GLFW_KEY_ESCAPE ) != GLFW_PRESS && glfwWindowShouldClose(window) == 0 );
        // Cleanup VAO, VBOs, and shaders from GPU
        cleanupDataFromGPU();
        // Close OpenGL window and terminate GLFW
        glfwTerminate();
        return 0;
}
```

P01, Exercise 1:
Graphics application to draw 2 triangles w/ different colors

```
void transferDataToGPUMemory(void)
{
		// VAO
		glGenVertexArrays(1, &VertexArrayID);
		glBindVertexArray(VertexArrayID);

		// Create and compile our GLSL program from the shaders
		programID = LoadShaders( "vertexshader.vs", "fragmentshader.fs" );
		// vertices for 2 triangles
		static const GLfloat g_vertex_buffer_data[] = {
				-1.0f, -1.0f, 0.0f,
				 1.0f, -1.0f, 0.0f,
				 0.0f,  1.0f, 0.0f,
				 -1.0f, 1.0f, 0.0f,
				 1.0f, 1.0f, 0.0f,
				 0.0f,  -1.0f, 0.0f,
		};

		// One color for each vertex
		static const GLfloat g_color_buffer_data[] = {
				1.0f,  0.0f,  0.0f,
				1.0f,  0.0f,  0.0f,
				1.0f,  0.0f,  0.0f,
				0.0f,  1.0f,  0.0f,
				0.0f,  1.0f,  0.0f,
				0.0f,  1.0f,  0.0f,
		};

		// Move vertex data to video memory; specifically to VBO called vertexbuffer
		glGenBuffers(1, &vertexbuffer);
		glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
		glBufferData(GL_ARRAY_BUFFER, sizeof(g_vertex_buffer_data), g_vertex_buffer_data, GL_STATIC_DRAW);
		// Move color data to video memory; specifically to CBO called colorbuffer
		glGenBuffers(1, &colorbuffer);
		glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
		glBufferData(GL_ARRAY_BUFFER, sizeof(g_color_buffer_data), g_color_buffer_data, GL_STATIC_DRAW);

}
```

P01, Exercise 1:
Graphics application to draw 2 triangles w/ different colors

```
void cleanupDataFromGPU()
{
	glDeleteBuffers(1, &vertexbuffer);
	glDeleteBuffers(1, &colorbuffer);
	glDeleteVertexArrays(1, &VertexArrayID);
	glDeleteProgram(programID);
}
```

P01, Exercise 1: Graphics application to draw 2 triangles w/ different colors

```
void draw (void)
{
	// Clear the screen
	glClear( GL_COLOR_BUFFER_BIT );
	// Use our shader
	glUseProgram(programID);

	// 1rst attribute buffer : vertices
	glEnableVertexAttribArray(0);
	glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
	glVertexAttribPointer(
		0,            // attribute 0. No particular reason for 0, but must match the layout in the shader.
		3,            // size
		GL_FLOAT,  // type
		GL_FALSE,   // normalized?
		0,            // stride
		(void*)0      // array buffer offset
	);

	// 2nd attribute buffer : colors
	glEnableVertexAttribArray(1);
	glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
	glVertexAttribPointer(
		1,            // attribute. No particular reason for 1, but must match the layout in the shader.
		3,            // size
		GL_FLOAT,   // type
		GL_FALSE,    // normalized?
		0,            // stride
		(void*)0      // array buffer offset
	);

	// Draw the 2 triangles !
	glDrawArrays(GL_TRIANGLES, 0, 6); // 6 indices starting at 0
	// Disable arrays of attributes for vertices
	glDisableVertexAttribArray(0);
	glDisableVertexAttribArray(1);
}
```

P01, Exercise 1: Graphics application to draw 2 triangles w/ different colors

vertexshader.vs

```
#version 330 core

// Input vertex data and color data
layout(location = 0) in vec3 vertexPosition;
layout(location = 1) in vec3 vertexColor;

// Output fragment data
out vec3 fragmentColor;

void main()
{
        // position of each vertex in homogeneous coordinates
        gl_Position.xyz = vertexPosition;
        gl_Position.w = 1.0;

        // the vertex shader just passes the color to fragment shader
        fragmentColor = vertexColor;
}
```

P01, Exercise 1:
Graphics application to draw 2 triangles w/ different colors

fragmentshader.fs

```
#version 330 core

// Interpolated values from the vertex shaders
in vec3 fragmentColor;

// Ouput data
out vec3 color;

void main()
{
	color = fragmentColor;
}
```

Summary

…:

- Scalar, point, and vector
- Vector space and affine space
- Basic point and vector operations
- Lines, planes, and triangles
- More generic geometric objects
- Rasterization
- Modern OpenGL pipeline