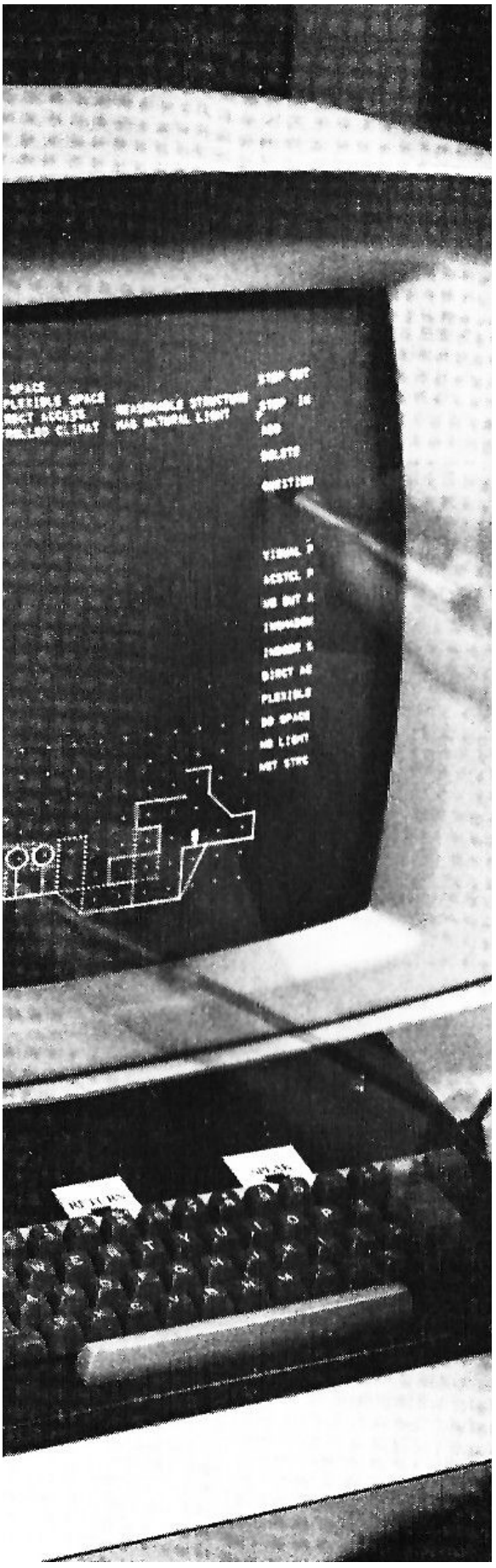
A” DS Aalto University
Design of Structures



Computational Design Concepts

compiled by **Luka Piškorec**
09.10.2018

Python programming - Learning resources

There are many online resources for learning Python. They are usually organized as step-by-step text or video tutorials. Some of them even offer a browser-based IDE (Integrated Development Environment), basically an editor that allows you to directly execute Python code (like Codecademy). Rhino 5 also comes with Python editor pre installed.

Codecademy

“Perfect for absolute beginners. Their free code tutorials are very popular, and for good reason. In this tutorial, you'll get a good idea of what it's like to code Python by having some free, hands-on lessons where you actually get to write some code.”

https://www.codecademy.com/learn/learn-python

Google's Python Class

“Great resource for someone looking to get a professional take on the learning the language. The class isn't for complete coding beginners, but those with a little coding knowledge will be able to learn Python using the videos, lectures, written materials, and code exercises that Google provides in this course.”

https://developers.google.com/edu/python/

A Byte of Python

“Free book on programming using the Python language. It serves as a tutorial or guide to the Python language for a beginner audience. If all you know about computers is how to save text files, then this is the book for you.”

Download PDF
https://www.gitbook.com/book/swaroopch/byte-of-python/details

Web version
https://python.swaroopch.com/

TutsPlus

“This TutsPlus tutorial provides you with dozens of different resources that can be used to learn Python, and it even gives you a syllabus so you can be sure to know what resource to use, and at which point in your coding journey you should use it. Perfect for both coding newbies and code veterans.”

https://code.tutsplus.com/articles/the-best-way-to-learn-python--net-26288

Plethora Project

“Plethora-Project.com is an initiative to accelerate computational literacy in the frame of architecture and design. It aligns with the "show me your screens" motto of the TopLap live-coding group attempting to get rid of Obscurantism in digital design. Since 2011, the Plethora Project has been posting over 200 video tutorials on vimeo and youtube. The videos are a contribution to the community that provided many of the open source tools available to the public. The tutorials cover issues of programing and computational design issues in a diverse range of software.”

Rhino - Python programming
https://www.plethora-project.com/education/2017/5/31/rhino-python-programming

Rhino - Nurbs modeling
https://www.plethora-project.com/education/2017/5/31/rhino-nurbs-modeling

Rhino - Grasshopper
https://www.plethora-project.com/education/2017/5/31/rhino-grasshopper

Python programming language

(referenced from Wikipedia)

On the origins of Python, it’s creator **Guido van Rossum** wrote in 1996:

> “...I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).”

The language's core philosophy is summarized in the document **The Zen of Python**, which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Python is a widely used **high-level programming language** for general-purpose programming, created by Guido van Rossum and first released in 1991. An **interpreted language**, Python has a design philosophy that emphasizes code readability (notably **using whitespace indentation to delimit code blocks** rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. Python features a **dynamic type system** and **automatic memory management**. It supports multiple programming paradigms, including **object-oriented**, **imperative**, **functional** and **procedural**, and has a large and comprehensive standard library.

A common neologism in the Python community is **pythonic**, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's **minimalist philosophy and emphasis on readability**. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.

Object-oriented programming (OOP)

Programming paradigm based on the concept of **"objects"**, which may contain data, in the form of fields, often known as **attributes**; and code, in the form of procedures, often known as **methods**. A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another. There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that **objects are instances of classes**, which typically also determine their type.

Imperative programming

Programming paradigm that uses **statements that change a program's state**. Computer program stores data in **variables**, which represent **storage locations in the computer's memory**. The contents of these memory locations, at any given point in the program's execution, is called the **program's state**. In much the same way that the imperative mood in natural languages expresses commands, an imperative program consists of commands for the computer to perform. **Imperative programming focuses on describing how a program operates.** The term is often used in contrast to declarative programming, which focuses on what the program should accomplish without specifying how the program should achieve the result.

Interpreted language

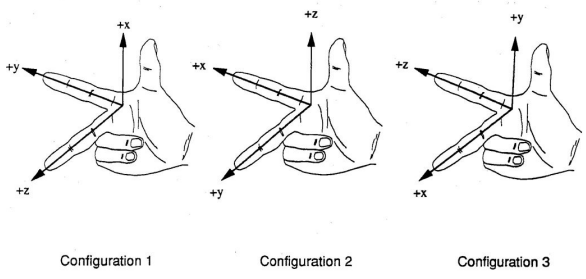
An interpreted language is a programming language for which most of its implementations **execute instructions directly, without previously compiling** a program into machine-language instructions. The interpreter executes the program directly, translating each statement into a sequence of one or more subroutines already compiled into machine code.

Coordinate systems and color spaces

(referenced from Wikipedia)

Right-hand rule

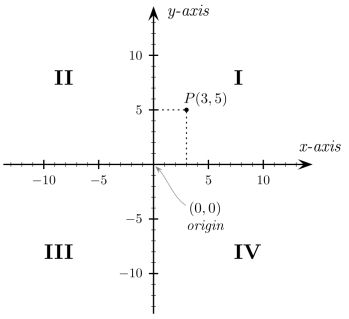
In mathematics and physics, the right-hand rule is a common mnemonic (a system such as a pattern of letters, ideas, or associations which assists in remembering something) for understanding **orientation conventions for vectors in three dimensions**. Most of the various left and right-hand rules arise from the fact that the three axes of 3-dimensional space have two possible orientations. This can be seen by holding your hands outward and together, palms up, with the fingers curled. If the curl of your fingers represents a movement from the first or X axis to the second or Y axis then the third or Z axis can point either along your left thumb or right thumb. Left and right-hand rules arise when dealing with coordinate axes, rotation, spirals, electromagnetic fields, mirror images and enantiomers in mathematics and chemistry.



Configuration 1 Configuration 2 Configuration 3

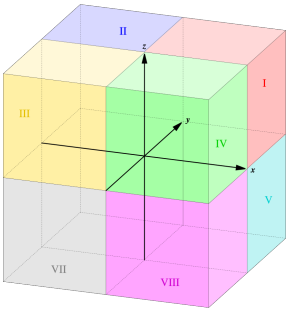
Note that the convention of assigning the index finger to the first axis (rather than the thumb) corresponds with the convention of finger-counting of the United Kingdom and United States, whereas **for Continental Europeans, the thumb represents the first digit to be counted**; the "natural" assignment of fingers to axes that leads to a "right"-handed rule would likewise differ in many other cultures.

Quadrant



The axes of a **two-dimensional coordinate system** divide the plane into **four infinite regions**, called quadrants, each bounded by two half-axes. These are often numbered from 1st to 4th and denoted by Roman numerals: I (where the signs of the (x,y) coordinates are (+,+), II (−,+), III (−,−), and IV (+,−). When the axes are drawn according to the mathematical custom, the numbering goes counter-clockwise starting from the upper right ("northeast") quadrant.

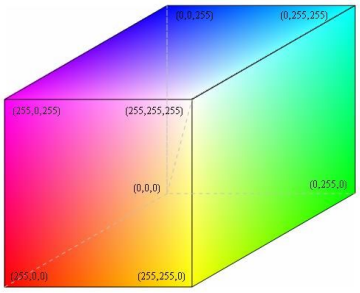
Octant



One of the **eight divisions** of a Euclidean **three-dimensional coordinate system** defined by the signs of the coordinates. It is similar to the two-dimensional quadrant and the one-dimensional ray. A convention for naming an octant is to give its list of signs, e.g. (+ - -) or (- + -). Octant (+ + +) is sometimes referred to as the first octant, although similar ordinal name descriptors are not defined for the other seven octants. The advantages of using the (+ - -) notation are its unambiguousness, and extensibility for higher dimensions.

Color space

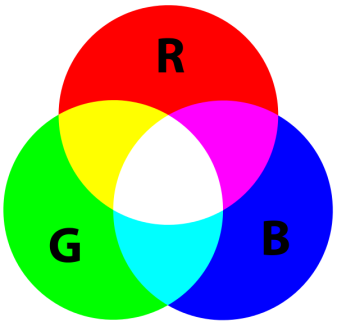
Specific **organization of colors**. In combination with physical device profiling, it allows for reproducible representations of color, in both analog and digital representations. A color space may be arbitrary, with particular colors assigned to a set of physical color swatches and corresponding assigned names or numbers (Pantone collection), or structured mathematically (Natural Color System and others). A color model is an **abstract mathematical model** describing the way colors can be represented as tuples of numbers, for example triples in **RGB** (red-green-blue) or quadruples in **CMYK** (cyan-magenta-yellow-black).



RGB color space represented as a 3D cube

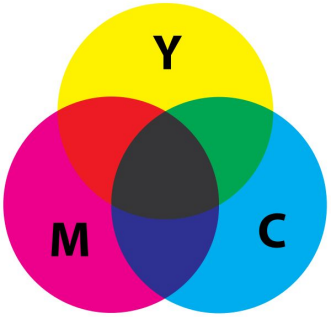
In 1802, **Thomas Young** postulated the existence of three types of photoreceptors (now known as cone cells) in the eye, each of which was sensitive to a particular range of visible light. **Hermann von Helmholtz** developed the Young–Helmholtz theory further in 1850 - that the **three types of cone photoreceptors** could be classified as short-preferring (blue), middle-preferring (green), and long-preferring (red), according to their response to the wavelengths of light striking the retina. The relative strengths of the **signals** detected by the three types of cones are **interpreted by the brain as a visible color**.

RGB color model



Additive color model in which **red, green and blue light** are added together in various ways to reproduce a broad array of colors. The main purpose of the RGB color model is for the sensing, representation and display of images in **electronic systems**, such as televisions and computers, though it has also been used in conventional photography.

CMYK color model



Subtractive color model, used in **color printing**, and is also used to describe the printing process itself. CMYK refers to the four inks used in some color printing: **cyan, magenta, yellow, and key (black)**.
The "K" in CMYK stands for key because in four-color printing, cyan, magenta, and yellow printing plates are carefully keyed, or aligned, with the key of the black key plate. Black is often used as outline and printed first.

Vectors

(referenced from Wikipedia)

Euclidean vectors

In mathematics, physics, and engineering, a Euclidean vector (sometimes called a geometric or spatial vector, or—as here—simply a vector) is a geometric object that has **magnitude** (or length) and **direction**. Vectors can be added to other vectors according to vector algebra. A Euclidean vector is frequently represented by a line segment with a definite direction, or graphically as an arrow, connecting an initial point A with a terminal point B and denoted by $\overrightarrow{AB}$

Vectors only contain information about **direction and magnitude**. In that sense, they don't have a fixed position in space and can be freely "moved" around. Vectors in three dimensions require three coordinates (vector components) to be uniquely defined. Point vectors are vectors which point from the origin of the coordinate system to the point with same coordinates as the components of the point-vector. This makes point-vector's and point's notation interchangeable.

Basic properties of vectors

Equality - Two vectors are said to be equal if they have the same magnitude and direction. Equivalently they will be equal if their coordinates are equal.

Opposite, parallel, and antiparallel vectors - Two vectors are opposite if they have the same magnitude but opposite direction. Two vectors are parallel if they have the same direction but not necessarily the same magnitude, or antiparallel if they have opposite direction but not necessarily the same magnitude.

Length - Each vector has a length which can be calculated using **Euclidean norm** formula:

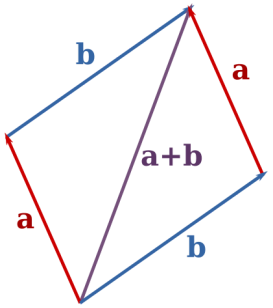
$$\|\mathbf{a}\| = \sqrt{{a_1}^2 + {a_2}^2 + {a_3}^2}$$

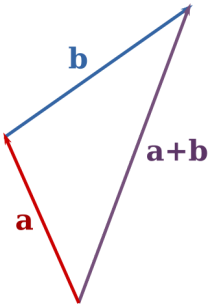
Euclidean norm works for vectors in any number of dimensions if we add squares of all components and calculate the square root of the sum. Calculating distances in higher-dimensional spaces has uses in modern data classification methods.

Unit vector - A unit vector is any vector with a **length of one**. Normally unit vectors are used simply to indicate direction. A vector of arbitrary length can be divided by its length to create a unit vector. This is known as **normalizing a vector**.

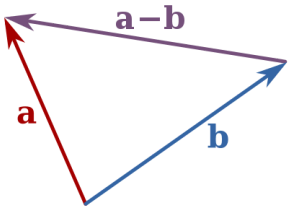
Zero vector - Vector with length zero. Unlike any other vector, it has an **arbitrary or indeterminate direction**, and cannot be normalized.

Vector addition - Vectors can be added to each other. Graphically, this is done by putting the start point of one on the end point of the other and drawing a vector from the start of the first to the end of the second vector. The result is always another **vector**.

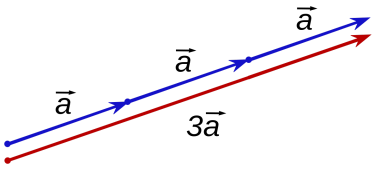




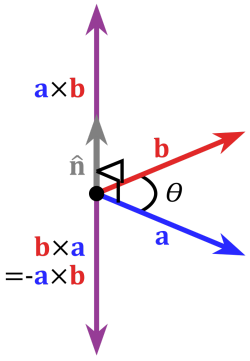
Vector subtraction - Vectors can be subtracted from each other. Graphically, one vector can be subtracted from the other by drawing a vector from the end of the second to the end of the first vector. The result is always another **vector**.



Scalar multiplication - Vector can be scaled in length by multiplying it with a skalar (a normal number, not another vector) in which case the direction stays the same. **If the scalar is negative, the vector will flip direction.** The result is always another **vector**.



Cross product - The cross product (also called the vector product or the **outer product**) of any two vectors is a **vector perpendicular to both of them** which completes a right-handed system. The right-handedness constraint is necessary because there exist two unit vectors that are perpendicular to any two vectors. Cross product can always be found unless two vectors have the same unit vectors or if one of them is a zero-vector. Otherwise, the result is always another **vector**.

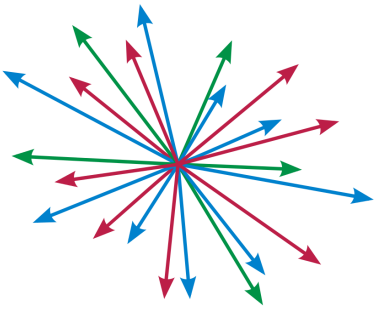


Dot product - The dot product of two vectors (sometimes called the **inner product**, or, since its **result is a scalar**, the scalar product) can be defined as the sum of the products of the components of each vector as:

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

Vector space

Collection of objects called **vectors**, which may be added together and multiplied ("scaled") by numbers, called **scalars**. Scalars are often taken to be real numbers, but there are also vector spaces with scalar multiplication by complex numbers, rational numbers, or generally any field. Euclidean vectors are an example of a vector space. They represent **physical quantities such as forces**: any two forces (of the same type) can be added to yield a third, and the multiplication of a force vector by a real multiplier is another force vector. In the same vein, but in a more geometric sense, **vectors representing displacements** in the plane or in three-dimensional space also form vector spaces. Vectors in vector spaces do not necessarily have to be arrow-like objects as they appear in the mentioned examples: vectors are regarded as abstract mathematical objects with particular properties, which in some cases can be visualized as arrows.



Possible representation of a six-dimensional vector space, where in addition to direction (three components) every vector also has a color (three components in RGB color space) totaling six numbers that uniquely describe a given vector

Vector spaces are the subject of **linear algebra** and are well characterized by their dimension, which, roughly speaking, specifies the number of **independent directions in the space**. Today, vector spaces are applied throughout mathematics, science and engineering, and furnish an abstract, **coordinate-free** way of dealing with geometrical and physical objects such as **tensors**.

Systems of randomness

(referenced from Wikipedia)

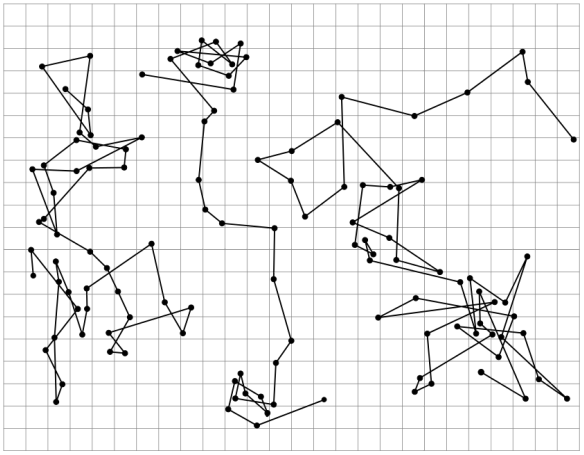
Random number generation

Generation of a sequence of numbers or symbols that cannot be reasonably predicted better than by a random chance, usually through a **hardware random-number generator** (RNG). Various applications of randomness have led to the development of several different methods for generating random data, of which some have existed since ancient times, among whose ranks are well-known "classic" examples, including the rolling of dice, coin flipping, the shuffling of playing cards, the use of yarrow stalks (for divination) in the I Ching, as well as countless other techniques. Because of the mechanical nature of these techniques, generating large numbers of sufficiently random numbers (important in statistics) required a lot of work and/or time. Thus, results would sometimes be collected and distributed as **random number tables**.

Random number generators have applications in gambling, statistical sampling, computer simulation, cryptography, completely randomized design, and other areas where producing an unpredictable result is desirable. Generally, in applications having unpredictability as the paramount, such as in security applications, hardware generators are generally preferred over **pseudo-random algorithms**, where feasible. Random number generators are very useful in developing **Monte Carlo method** simulations, as debugging is facilitated by the ability to run the same sequence of random numbers again by starting from the same random seed. They are also used in cryptography – so long as the seed is secret. Sender and receiver can generate the same set of numbers automatically to use as keys. Weaker forms of randomness are used in **hash algorithms** and in creating **amortized searching** (method for analyzing a given algorithm's time complexity, or how much of a resource, especially time or memory, it takes to execute) and **sorting algorithms**.

Brownian motion

Random motion of particles suspended in a fluid (a liquid or a gas) resulting from their collision with the fast-moving atoms or molecules in the gas or liquid.
This transport phenomenon is named after the botanist **Robert Brown**. In 1827, while looking through a microscope at particles trapped in cavities inside pollen grains in water, he noted that the particles moved through the water; but he was not able to determine the mechanisms that caused this motion. Atoms and molecules had long been theorized as the constituents of matter, and **Albert Einstein** published a paper in 1905 that **explained in precise detail** how the motion that Brown had observed was a result of the **pollen being moved by individual water molecules**. This explanation of Brownian motion served as convincing evidence that atoms and molecules exist, and was further verified experimentally by Jean Perrin in 1908.



From the book of Jean Baptiste Perrin, Les Atomes

The direction of the force of atomic bombardment is constantly changing, and at different times the particle is hit more on one side than another, leading to the seemingly **random nature of the motion**. Brownian motion is among the simplest of the **continuous-time stochastic (or probabilistic) processes**, and it is a limit of both simpler and more complicated stochastic processes (see random walk and Donsker's theorem). This universality is closely related to the universality of the **normal distribution**.

Monte Carlo method, voxels, cellular automata and transformation matrices
(referenced from Wikipedia)

Monte Carlo method

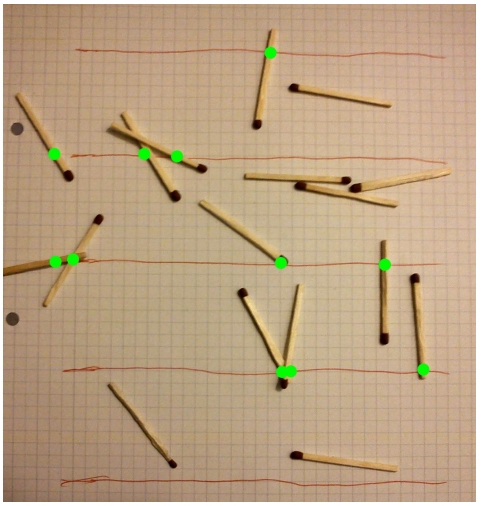
A broad class of computational algorithms that rely on repeated **random sampling** to obtain numerical results. Their essential idea is using randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches. Monte Carlo methods are mainly used in three distinct problem classes: optimization, numerical integration, and generating draws from a probability distribution. In physics-related problems, Monte Carlo methods are useful for **simulating systems** with many coupled degrees of freedom, such as fluids, disordered materials, strongly coupled solids, and cellular structures.

In principle, Monte Carlo methods can be used to solve any problem having a probabilistic interpretation. By the **law of large numbers**, integrals described by the expected value of some random variable can be approximated by taking the empirical mean of independent samples of the variable. Monte Carlo methods vary, but tend to follow a particular pattern:

1. **Define** a domain of possible inputs
2. **Generate** inputs randomly from a probability distribution over the domain
3. **Perform** a deterministic computation on the inputs
4. **Aggregate** the results

Uses of Monte Carlo methods **require large amounts of random numbers**, and it was their use that spurred the development of pseudorandom number generators, which were far quicker to use than the tables of random numbers that had been previously used for statistical sampling.

An early variant of the Monte Carlo method can be seen in the **Buffon's needle** experiment, in which π can be estimated by dropping needles on a floor made of parallel and equidistant strips. The modern version of the **Markov Chain** Monte Carlo method was invented in the late 1940s by **Stanislaw Ulam**, while he was working on nuclear weapons projects at the Los Alamos National Laboratory. Immediately after Ulam's breakthrough, **John von Neumann** understood its importance and programmed the ENIAC computer to carry out Monte Carlo calculations.



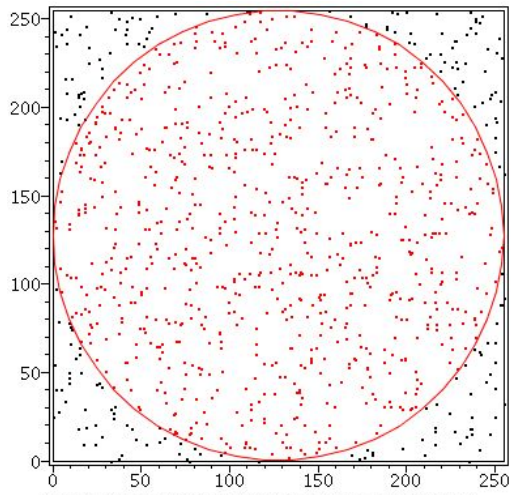
A example of Buffon's needle experiment to estimate π, yielding the value of 34/11 ≈ 3.1 based on 17 throws

Markov process, named after the Russian mathematician Andrey Markov, is a stochastic process that satisfies the **Markov property** (sometimes characterized as **"memorylessness"**). Roughly speaking, a process satisfies the Markov property if one can make predictions for the future of the process based solely on its present state. In such a process, **future and past states are independent**. This is a necessary requirement for Monte Carlo method. A simple **coin toss** is an example of such a Markov process, as is **Brownian motion**, due to the fact that the displacement of the particle does not depend on its past displacements.

Approximating the value of π with Monte Carlo

Consider a circle inscribed in a unit square. Given that the circle and the square have a ratio of areas that is π/4, the value of π can be approximated using a Monte Carlo method:

1. Draw a square, then inscribe a circle within it
2. Uniformly scatter objects of uniform size over the square
3. Count the number of objects inside the circle and the total number of objects
4. The ratio of the inside-count and the total-sample-count is an estimate of the ratio of the two areas, which is π/4. Multiply the result by 4 to estimate π

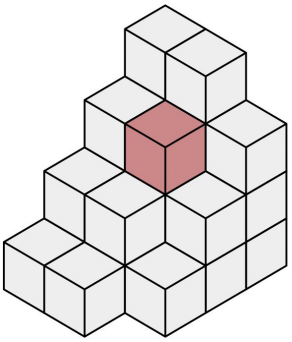


1000 point Monte Carlo approximation of π = 3.1960

In this procedure the domain of inputs is the square that circumscribes the circle. We generate random inputs by scattering grains over the square then perform a computation on each input (test whether it falls within the circle). Finally, we aggregate the results to obtain our final result, the approximation of π. There are two important points: Firstly, if the grains are not **uniformly distributed**, then the approximation will be poor. Secondly, there should be a **large number of inputs**. The approximation is generally poor if only a few grains are randomly dropped into the whole square. On average, the approximation improves as more grains are dropped.

Voxels

A **value** on a regular grid in **three-dimensional space**. The word voxel originated by analogy with the word "pixel", with vo representing **"volume"** and el representing **"element"**. As with pixels in a bitmap, voxels themselves do not typically have their position (their coordinates) explicitly encoded along with their values. Instead, rendering systems infer the position of a voxel based upon its **position relative to other voxels** (i.e., its position in the data structure that makes up a single volumetric image).



A voxel represents a value on a regular grid in three-dimensional space

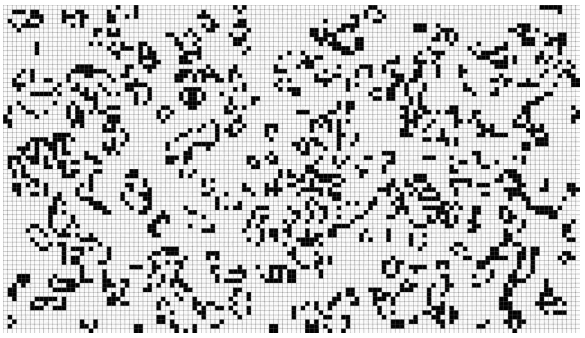
In contrast to pixels and voxels, points and polygons are often explicitly represented by the coordinates of their vertices. A direct consequence of this difference is that polygons can efficiently represent simple 3D structures with lots of empty or homogeneously filled space, while voxels excel at representing **regularly sampled spaces** that are **non-homogeneously filled**. Common uses of voxels include volumetric imaging in medicine and representation of terrain in games and simulations.

Voxel data point can consist of a single piece of data, such as an opacity, or multiple pieces of data, such as a color in addition to opacity. A voxel represents only a single point on this grid, not a volume; the **space between each voxel is not represented in a voxel-based dataset**. Depending on the type of data and the intended use for the dataset, this missing information may be reconstructed and/or approximated, e.g. via interpolation.

Cellular automaton

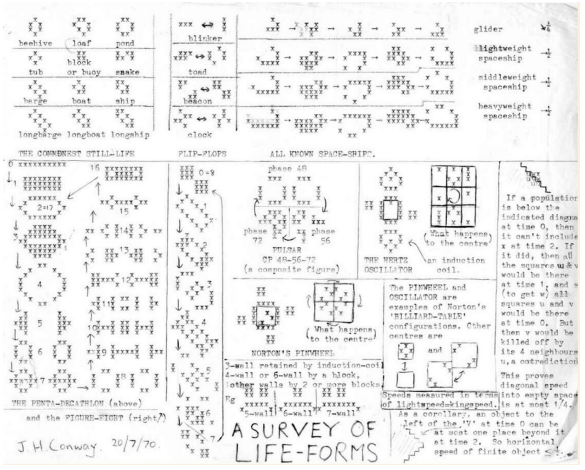
A **discrete model** studied in computer science, mathematics, physics, complexity science, theoretical biology and microstructure modeling. Cellular automata are also called cellular spaces, tessellation automata, homogeneous structures, cellular structures, tessellation structures, and iterative arrays.

A cellular automaton consists of a **regular grid of cells**, each in one of a **finite number of states**, such as on and off. The grid can be in any finite number of dimensions. For each cell, a set of cells called its **neighborhood** is defined relative to the specified cell. An initial state (time t = 0) is selected by assigning a state for each cell. A new generation is created (advancing t by 1), according to some fixed rule that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. Typically, the **rule for updating the state of cells** is the same for each cell and does not change over time, and is applied to the whole grid simultaneously.



The Game of Life is a cellular automaton devised by the British mathematician John Horton Conway in 1970

The concept was originally discovered in the 1940s by **Stanislaw Ulam** and **John von Neumann** while they were contemporaries at Los Alamos National Laboratory. While studied by some throughout the 1950s and 1960s, it was not until the 1970s and **Conway's Game of Life**, a two-dimensional cellular automaton, that interest in the subject expanded beyond academia.



Taxonomy of Life creatures, a rule set for Conway's Game of Life from 1970

In the 1970s a two-state, two-dimensional cellular automaton named **Game of Life** became widely known, particularly among the early computing community. Invented by **John Conway**, its rules are as follows:

1. **n < 2, live** - any live cell with fewer than two live neighbours dies, as if caused by underpopulation.
2. **n = 2 or 3, live** - any live cell with two or three live neighbours lives on to the next generation.
3. **n > 3, live** - any live cell with more than three live neighbours dies, as if by overpopulation.
4. **n = 3, dead** - any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

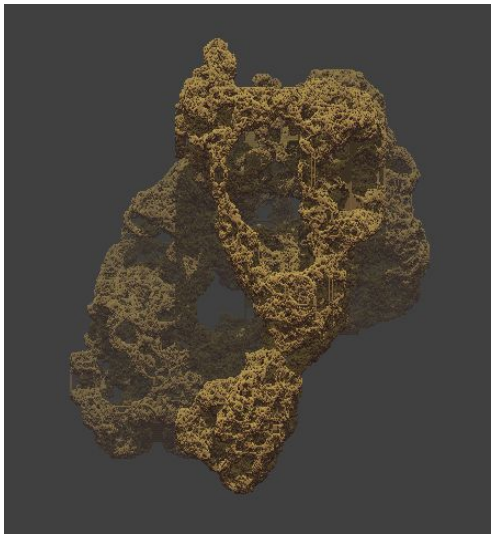
Despite its simplicity, the system achieves an impressive diversity of behavior, **fluctuating between apparent randomness and order**. One of the most apparent features of the Game of Life is the frequent occurrence of gliders, arrangements of cells that essentially move themselves across the grid. It is possible to arrange the automaton so that the gliders interact to perform computations, and after much effort it has been shown that the Game of Life can emulate a **universal Turing machine**.

In the 1980s, **Stephen Wolfram** engaged in a systematic study of one-dimensional cellular automata, or what he calls elementary cellular automata; his research assistant Matthew Cook showed that one of these rules is Turing-complete.

Wolfram published **A New Kind of Science** in 2002, claiming that cellular automata have applications in many fields of science like computer processors and cryptography. The primary **classifications of cellular automata**, as outlined by Wolfram, are numbered one to four:

1. Automata in which patterns generally stabilize into **homogeneity**
2. Automata in which patterns evolve into mostly **stable or oscillating** structures
3. Automata in which patterns evolve in a seemingly **chaotic** fashion
4. Automata in which patterns become extremely **complex** and may last for a long time, with stable local structures

This last class are thought to be computationally universal, or capable of simulating a **Turing machine**. Cellular automata can **simulate** a variety of **real-world systems**, including biological and chemical ones.



Model resembling an organic structure, generated with cellular automata and high-resolution voxels by Tom Lowe

Cellular automata smoothing algorithm

In **game design**, there is an old and fairly well documented trick to use cellular automata to generate **cave-like structures**. The basic idea is to fill the first map randomly, then repeatedly create new maps using the **4-5 rule**:

1. A tile becomes a wall if it already **was a wall** in previous iteration and **4 or more of its eight neighbors were walls**
2. A tile becomes a wall if it **was not a wall** in previous iteration and **5 or more of its neighbors were walls**

Put more succinctly, a tile is a wall if the 3x3 region centered on it contained at least 5 walls. Each iteration makes each tile more like its neighbors, and the amount of overall "noise" is gradually reduced.

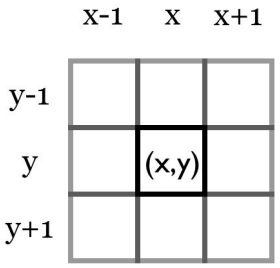
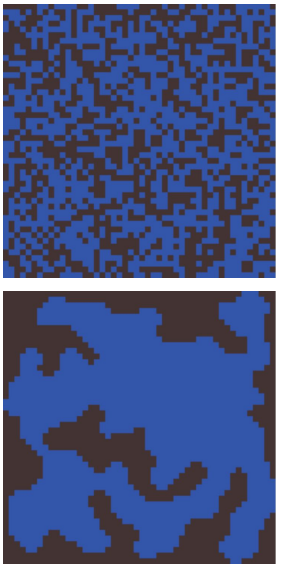


Diagram which shows the relation of every cell to its neighbours with generalized 2D grid coordinates



Randomly distributed points on the grid (left) and the cave after six cellular automaton simulation steps (right)

Transformation matrix

Matrix is a **rectangular array** of numbers, symbols, or expressions, arranged in rows and columns. **Transformation matrices** are mathematical models used for **moving, rotating, projecting, and scaling objects in computer graphics**. Matrices are also used for transformations between coordinate systems, for example from the 3D world coordinate to the 2D screen coordinate system. Basic operations on matrices are:

Addition - provided that they have the same size (each matrix has the same number of rows and the same number of columns as the other), two matrices can be added or subtracted element by element.

Multiplication - two matrices can be multiplied only when the number of columns in the first equals the number of rows in the second. Any matrix can be multiplied element-wise by a scalar from its associated field.

A major application of matrices is to **represent linear transformations**, that is, generalizations of linear functions such as f(x) = 4x. The **rotation of vectors** in three-dimensional space is a linear transformation, which can be represented by a **rotation matrix R**. If **v** is a **column vector** (a matrix with only one column) describing the position of a point in space, the **product Rv** is a column vector describing the position of that **point after a rotation**.

Matrix Multiplication

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ q \end{bmatrix}$$

We can transform a vector (or a point, the process is the same) by writing it in a column vector notation and multiplying it with a transformation matrix. The result is the transformed vector.

The **product of two transformation matrices** is a matrix that represents the **composition of two transformations**. If A and B are the matrices of two linear transformations, then the effect of applying first A and then B to an object (vector or point) is simply the product of the individual matrices. A consequence of the ability to compose transformations by multiplying their matrices is that **transformations can also be inverted** by simply **inverting their matrices**. So, A^{-1} represents the transformation that "undoes" A.

X-Rotation in 3D

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

Z-Rotation in 3D

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

Y-Rotation in 3D

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

Translation in 3D

$$\begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4×4 transformation matrices are widely used in 3D computer graphics. Graphic chips (GPUs) are developed and optimized for performing matrix calculations fast and in parallel.

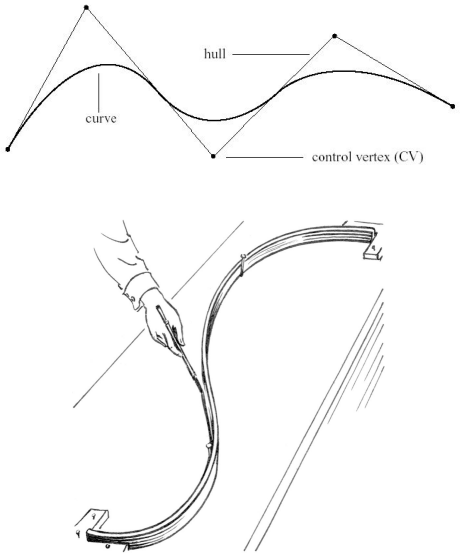
Transformation matrices used in computer graphics are called, depending on their application, **affine transformation** matrices, **projective transformation** matrices, or more generally **non-linear transformation matrices**. Except in **computer graphics**, applications of matrices are found in most **scientific fields**. In every branch of physics, including classical mechanics, optics, electromagnetism, quantum mechanics, and quantum electrodynamics, they are used to study physical phenomena, such as the motion of rigid bodies.

Curve, surface and solid modeling

(referenced from Wikipedia)

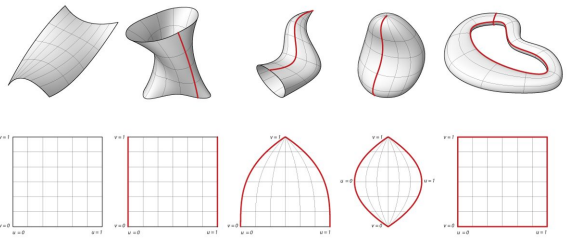
NURBS

Non-uniform rational basis spline is a mathematical model commonly used in computer graphics for generating and **representing curves and surfaces**. It offers great flexibility and precision for handling both analytic (surfaces defined by common mathematical formulae) and modeled shapes. NURBS are commonly used in computer-aided design (CAD), manufacturing (CAM), and engineering (CAE). NURBS tools are also found in various 3D modeling and animation software packages.



A spline curve and its physical counterpart

They can be efficiently handled by the computer programs and yet allow for easy human interaction. NURBS surfaces are functions of two parameters mapping to a **surface in three-dimensional space**. The shape of the surface is determined by control points. NURBS surfaces can represent, in a compact form, simple geometrical shapes. In general, editing NURBS curves and surfaces is highly intuitive and predictable. Control points are always either connected directly to the curve/surface, or act as if they were connected by a rubber band.



Examples of NURBS surfaces and their seam lines

Before computers, designs were drawn by hand on paper with various drafting tools. Rulers were used for straight lines, compasses for circles, and protractors for angles. But many shapes, such as the **freeform curve of a ship's bow**, could not be drawn with these tools. Although such curves could be drawn freehand at the drafting board, shipbuilders often needed a life-size version which could not be done by hand. Such large drawings were done with the help of flexible strips of wood, called **splines**. The splines were held in place at a number of predetermined points, called **"ducks"**; between the ducks, the elasticity of the spline material caused the strip to take the shape that minimized the energy of bending, thus creating the **smoothest possible shape that fit the constraints**. The shape could be tweaked by moving the ducks.

In 1946, mathematicians started studying the spline shape, and derived the piecewise **polynomial formula** known as the spline curve or **spline function**. I. J. Schoenberg gave the spline function its name after its resemblance to the mechanical spline used by draftsmen. As computers were introduced into the design process, the physical properties of such splines were investigated so that they could be modelled with mathematical precision and reproduced where needed. Pioneering work was done in France by **Renault engineer Pierre Bézier**, and **Citroën's physicist and mathematician Paul de Casteljau**. They worked nearly parallel to each other, but because Bézier published the results of his work, **Bézier curves** were named after him, while de Casteljau’s name is only associated with related algorithms.

BREP

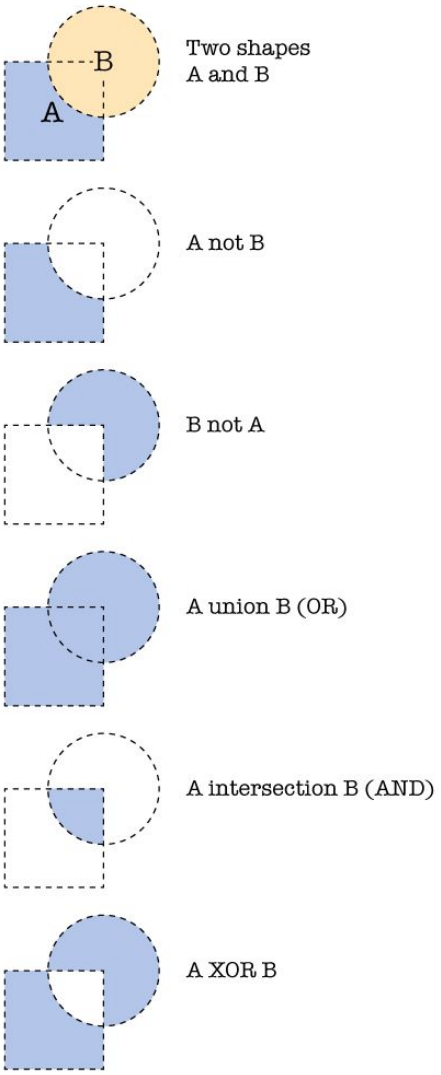
Boundary representation is a method for representing shapes using the limits. A solid is represented as a **collection of connected surface elements**, the boundary between solid and nonsolid. Boundary representation of models are composed of two parts: topology and geometry (surfaces, curves and points). The main topological items are: **faces, edges and vertices**. A face is a bounded portion of a surface; an edge is a bounded piece of a curve and a vertex lies at a point. Compared to the constructive solid geometry (CSG) representation, which uses only primitive objects and Boolean operations to combine them, boundary representation is more flexible and has a much richer operation set. In addition to the **Boolean operations**, B-rep has extrusion (or sweeping), chamfer, blending, drafting, shelling, tweaking and other operations which make use of these.

CSG

Constructive solid geometry is a technique used in solid modeling. It allows a modeler to create a complex surface or object by using **Boolean operators** to combine simpler objects. Often CSG presents a model or surface that appears visually complex, but is actually made out of cleverly combined or de-combined objects. In 3D computer graphics and CAD, CSG is often used in procedural modeling.

The simplest solid objects used for the representation are called **primitives**. Typically they are the objects of simple shape: **cuboids, cylinders, prisms, pyramids, spheres, cones**. The set of allowable primitives is limited by each software package. It is said that an object is constructed from primitives by means of allowable operations, which are typically **Boolean operations** on sets: **union, intersection and difference**, as well as geometric transformations of those sets. A primitive can typically be described by a procedure which accepts some number of parameters.

Constructive solid geometry has a number of practical uses. It is used in cases where simple geometric objects are desired, or where mathematical accuracy is important. One of the advantages of CSG is that it can easily assure that objects are **"solid"** or **water-tight** if all of the primitive shapes are water-tight. This can be important for some manufacturing or engineering computation applications.



Different boolean operations on solid geometry

Geometry curvature, particles and inverse-square law

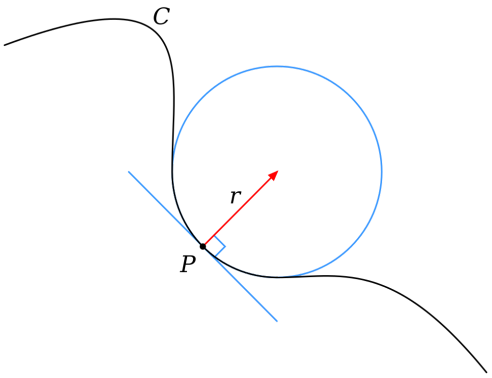
(referenced from Wikipedia)

Curvature

In mathematics, curvature is any of a number of loosely related concepts in different areas of geometry. Intuitively, curvature is the **amount** by which a **geometric object** such as a surface **deviates from being** a **flat** plane, or a curve from **being straight** as in the case of a line, but this is defined in different ways depending on the context.

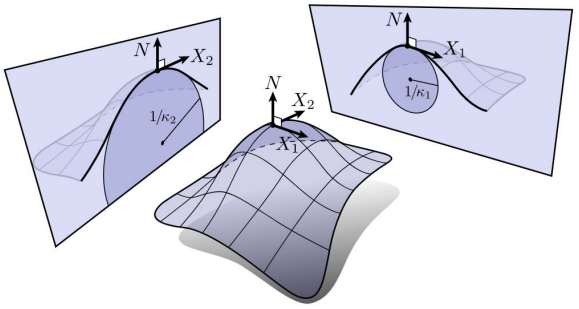
Extrinsic curvature

Relates to the radius of circles that touch the object. It is not detectable to someone who can't study the three-dimensional space surrounding the surface on which he resides. The canonical example of **extrinsic curvature** is that of a circle, which has a **curvature equal to the reciprocal of its radius** everywhere. Smaller circles bend more sharply, and hence have higher curvature.



The curvature of a smooth curve is defined as the curvature of its osculating circle at each point. The curvature of a circle is defined to be the reciprocal of the radius: ***K = 1/r***

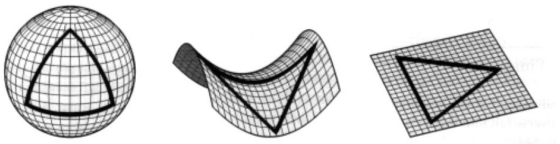
Curvature is **normally a scalar quantity**, but one may also define a curvature vector that takes into account the direction of the bend in addition to its magnitude.
At each point of a **differentiable surface** in 3-dimensional Euclidean space one may choose a unit **normal vector**. A **normal plane** at the point is one that contains the normal vector, and will therefore also contain a unique **direction tangent** to the surface and cut the surface in a plane curve, called **normal section**. This curve will in general have different curvatures for different normal planes at the point. **Two principal curvatures** at the point are the **maximum** and **minimum values** of this curvature.



Example of two principal curvatures with their corresponding normal section planes

Intrinsic curvature

Defined in terms of the lengths of curves within a Riemannian manifold. It is detectable to the "inhabitants" of a surface and not just outside observers. It is an **intrinsic** measure of **curvature**, depending only on distances that are measured on the surface, not on the way it is isometrically embedded in any space. **Gauss curvature** is a **product of the principal curvatures** at the given point.



Examples of positive, negative and flat curvature.

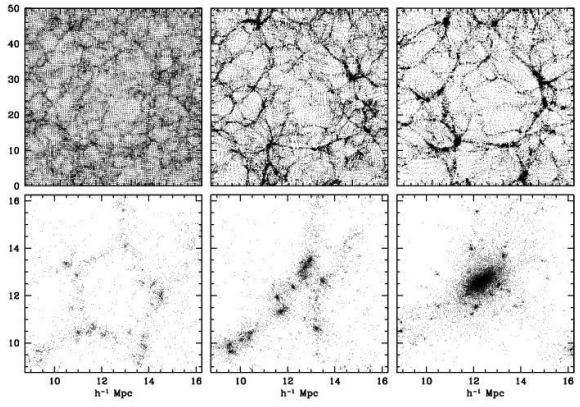
For example, a sphere of radius r has Gaussian curvature $1/r^2$ everywhere, and a **flat plane** and a **cylinder** have **Gaussian curvature 0 everywhere**. The Gaussian curvature can also be negative, as in the case of a hyperboloid or the inside of a torus.

Particles

Small **localized object** (corpuscle in older texts) to which can be ascribed several physical or chemical properties such as volume or mass. They vary greatly in size or quantity, from **subatomic particles** like the electron, to microscopic particles like **atoms** and **molecules**, to macroscopic particles like **powders** and other **granular materials**. Particles can also be used to create scientific models of even larger objects depending on their density, such as **humans moving in a crowd** or **celestial bodies in motion**. The concept of particles is particularly useful when **modelling nature**, as the full treatment of many phenomena can be complex and also involve difficult computation. The treatment of large numbers of particles is the **realm of statistical physics**.

N-body simulations

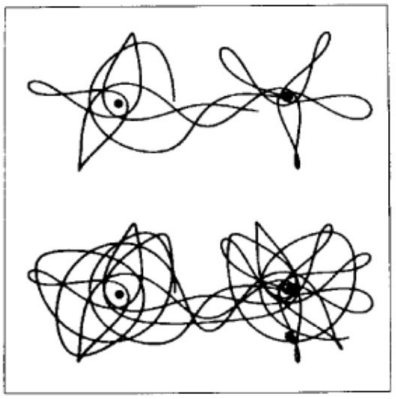
Also called N-particle simulations, are simulations of **dynamical systems of particles** under the influence of certain conditions, such as being subject to gravity. These simulations are very common in cosmology and computational fluid dynamics. N refers to the number of particles considered. As simulations with higher N are more computationally intensive, systems with large numbers of actual particles will often be approximated to a smaller number of particles, and simulation algorithms need to be optimized through various methods.



From “A hierarchy of voids: Much ado about nothing” paper by Ravi K. Sheth and Rien van de Weygaert.

Three-body problem

Problem of taking an initial set of data that specifies the positions, masses, and velocities of three bodies for some particular point in time and then **determining the motions of the three bodies**, in accordance with Newton's laws of motion and of universal gravitation which are the laws of classical mechanics. The three-body problem is a special case of the **n-body problem**. Unlike two-body problems, there is no general closed-form solution for every condition and **numerical methods are needed** to solve these problems. Historically, the first specific three-body problem to receive extended study was the one involving the Moon, the Earth, and the Sun.



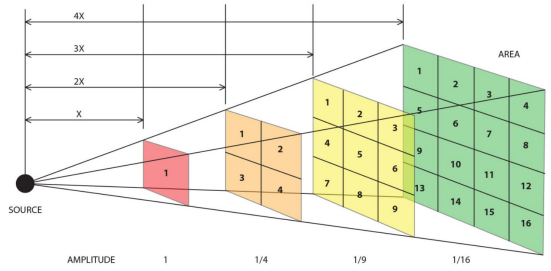
Orbits related to the motion of three bodies that exert gravitational attraction between each other.

Among classical physical systems, the **n-body problem** usually refers to a galaxy or to a cluster of galaxies; planetary systems, such as stars, planets, and their satellites, can also be treated as n-body systems. Some applications are conveniently treated by **perturbation theory**, in which the system is considered as a two-body problem plus additional forces causing deviations from a hypothetical unperturbed two-body trajectory.

In work summarized in 1892–1899, **Henri Poincaré** established the existence of an **infinite number of periodic solutions** to the restricted three-body problem, and established techniques for continuing these solutions into the general three-body problem.

Inverse-square law

Any **physical law** stating that a specified physical quantity or intensity is **inversely proportional to the square of the distance** from the source of that physical quantity.



The fundamental cause for the inverse-square law can be understood as geometric dilution corresponding to point-source radiation into three-dimensional space.

The inverse-square law generally applies when some force, energy, or other conserved quantity is **evenly radiated outward from a point source** in three-dimensional space. Since the surface area of a sphere (which is $4\pi r^2$) is proportional to the square of the radius, as the emitted radiation gets farther from the source, it is spread out over an area that is increasing in proportion to the square of the distance from the source. Hence, the intensity of radiation passing through any unit area (directly facing the point source) is inversely proportional to the square of the distance from the point source. **Gauss' law** is similarly applicable, and can be used with any physical quantity that acts in accord to the inverse-square relationship.

Gravity

Attraction of two objects with mass. **Newton's law** states: The gravitational attraction force between two point masses is directly proportional to the product of their masses and inversely proportional to the square of their separation distance. The force is always attractive and acts along the line joining them.

Electrostatics

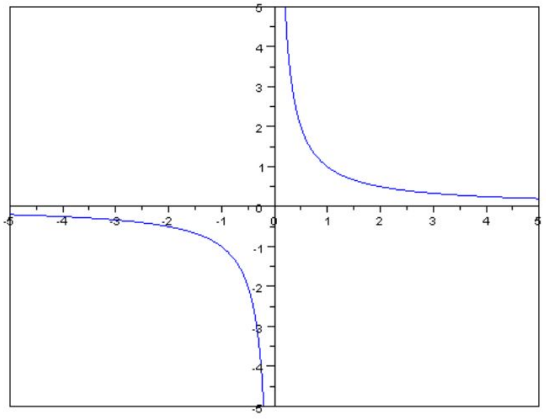
The force of attraction or repulsion between two electrically charged particles, in addition to being directly proportional to the product of the electric charges, is inversely proportional to the square of the distance between them. This is known as **Coulomb's law**.

Light

The intensity (or illuminance or irradiance) of light or other linear waves radiating from a point source (energy per unit of area perpendicular to the source) is inversely proportional to the square of the distance from the source. An object (of the same size) twice as far away, receives only one-quarter the energy (in the same time period).

Acoustics

The sound pressure of a spherical wavefront radiating from a point source decreases by 50% as the distance is doubled. Measured in dB, the decrease is still 6.02 dB, since dB represents an intensity ratio. The pressure ratio (as opposed to power ratio) is not inverse-square, but is **inverse-proportional** (inverse distance law).



Inverse proportionality diagram. Variable y is directly proportional to the variable x.

Recursion, L-systems and fractals

(referenced from Wikipedia)

Recursion

Occurs **when a thing is defined in terms of itself** or of its type. Recursion is used in a variety of disciplines ranging from linguistics to logic. The most common application of recursion is in mathematics and computer science, where a **function being defined is applied within its own definition**.

Ouroboros, an ancient symbol depicting a serpent or dragon eating its own tail.

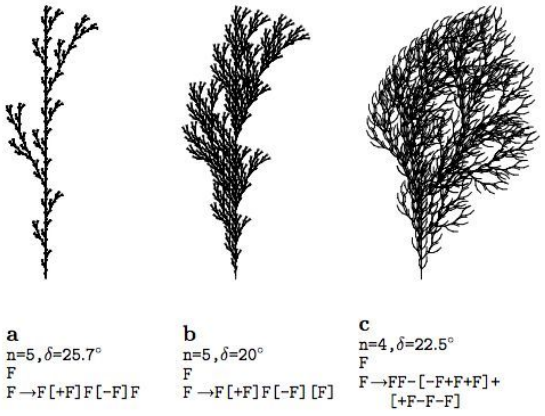
Recursion in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem (as **opposed to iteration**). The approach can be applied to many types of problems, and recursion is **one of the central ideas of computer science**. Niklaus Wirth, in his book *Algorithms + Data Structures = Programs* from 1976, wrote:

> *"The power of recursion evidently lies in the possibility of defining an infinite set of objects by a finite statement."*

Most computer programming languages support recursion by **allowing a function to call itself** within the program text. Some functional programming languages do not define any looping constructs but rely solely on recursion to repeatedly call code.

Lindenmayer system

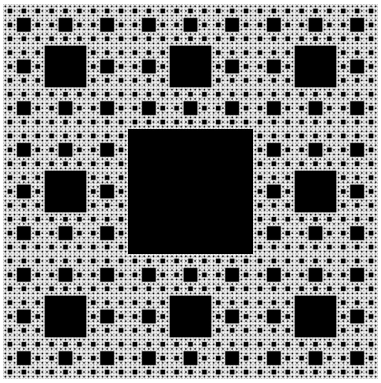
Parallel rewriting system and a **type of formal grammar**. An L-system consists of an alphabet of symbols that can be used to make strings, a collection of production rules that expand each symbol into some larger string of symbols, an initial "**axiom**" string from which to begin construction, and a mechanism for translating the generated strings into **geometric structures**. L-systems were introduced and developed in 1968 by **Aristid Lindenmayer**, a Hungarian theoretical biologist and botanist at the University of Utrecht. Lindenmayer used L-systems to describe the behaviour of plant cells and to model the growth processes of plant development. L-systems have also been used to model the morphology of a variety of organisms and can be used to **generate self-similar fractals** such as iterated function systems.



The **recursive nature** of the L-system rules leads to self-similarity and thereby, fractal-like forms are easy to describe with an L-system. Plant models and natural-looking organic forms are easy to define, as by increasing the recursion level the form slowly “grows” and becomes more complex. Lindenmayer systems are also popular in the **generation of artificial life**.

Fractals

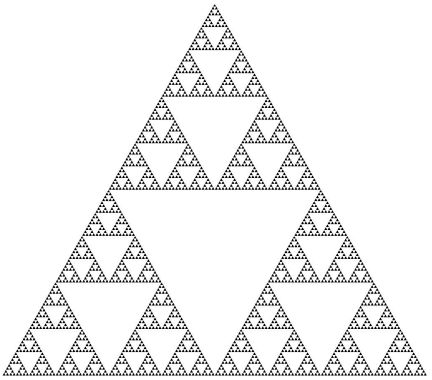
In mathematics, an **abstract object** used to **describe and simulate naturally occurring objects**. Artificially created fractals commonly exhibit similar patterns at increasingly small scales. It is also known as expanding symmetry or **evolving symmetry**. If the replication is exactly the same at every scale, it is called a self-similar pattern. Fractals also include the idea of a **detailed pattern that repeats itself**.



Sierpinski carpet (6 iterations), a two-dimensional fractal first described by Wacław Sierpiński in 1916

Fractals are different from other geometric figures because of the way in which they scale. Doubling the edge lengths of a polygon multiplies its area by four, which is two (the scale ratio) raised to the power of two (the dimension of the space the polygon resides in). Likewise, if the radius of a sphere is doubled, its volume scales by eight, which is two (the scale ratio) to the power of three (the dimension that the sphere resides in). But if a fractal's one-dimensional lengths are all doubled, the spatial content of the fractal scales by a power that is not necessarily an integer. This power is called the **fractal dimension of the fractal**, and it usually exceeds the fractal's topological dimension. As mathematical equations, fractals are usually **nowhere differentiable** (in a concrete sense, this means fractals cannot be measured in traditional ways). An infinite fractal curve can be conceived of as winding through space differently from an ordinary line, still being a 1-dimensional line yet having a fractal dimension indicating it also resembles a surface.
The mathematical roots of the idea of fractals have been traced throughout the years as a formal path of published works, starting in the 17th century with **notions of recursion**, then moving through increasingly rigorous mathematical treatment of the concept to the study of continuous but not differentiable functions in the 19th century by the seminal work of Bernard Bolzano, Bernhard Riemann, and Karl Weierstrass, and on to the coining of the word fractal in the 20th century with a subsequent burgeoning of interest in fractals and **computer-based modelling in the 20th century**. The term "fractal" was first used by mathematician **Benoit Mandelbrot** in 1975. Mandelbrot based it on the Latin **frāctus meaning "broken" or "fractured"**, and used it to extend the concept of theoretical fractional dimensions to geometric patterns in nature.

Fractals are not limited to geometric patterns, but **can also describe processes in time**. Fractal patterns with various degrees of self-similarity have been rendered or studied in images, structures and sounds and found in nature, technology, art, and law. Fractals are of particular relevance in the field of **chaos theory**, since the graphs of most chaotic processes are fractal.



Sierpinski triangle (7 iterations), is a fractal resulting from doing the following: 1. Start with an equilateral triangle. 2. Remove center part. 3. Do the same for the three largest equilateral triangles left in this one.

Dynamic relaxation and minimal surfaces
(referenced from Wikipedia)

Dynamic relaxation

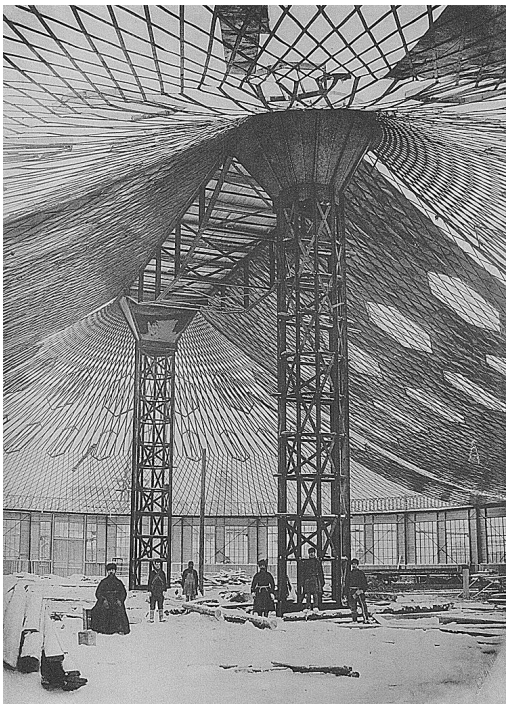
A numerical method which can be used for **form-finding** for cable and fabric structures. The aim is to find a geometry where **all forces are in equilibrium**. In the past this was done by direct modelling, using hanging chains and weights (Antoni Gaudi used it while designing Sagrada Familia), or by using soap films, which have the property of adjusting to find a **minimal surface**.

The dynamic relaxation method is based on **discretizing the continuum** under consideration by lumping the **mass at nodes** and defining the relationship between nodes in terms of **stiffness**. The system oscillates about the equilibrium position under the influence of loads. An **iterative process** is followed by simulating a pseudo-dynamic process in time, with each iteration based on an update of the geometry.

Iteration steps for calculating static equilibrium are:

1. **set the initial kinetic energy** and all nodal velocity components to zero
2. **compute** the geometry set and the **applied load component** for each node
3. compute the **residual force** for each node
4. reset the residual forces of **constrained nodes** to zero
5. **update velocity and coordinates** of the nodes
6. return to step 3 until the structure is in **static equilibrium**

Dynamic relaxation is an example of **optimization**, in which parameter which is being optimized for is the sum of all forces between the nodes. The goal of the optimization is to reduce that sum to zero exactly or to the best possible approximation. This is achieved by **changing the distances and angles** between the nodes in a way which **minimizes the total sum of forces**.



The world's first tensile steel shell by Vladimir Shukhov (during construction), Nizhny Novgorod, 1895

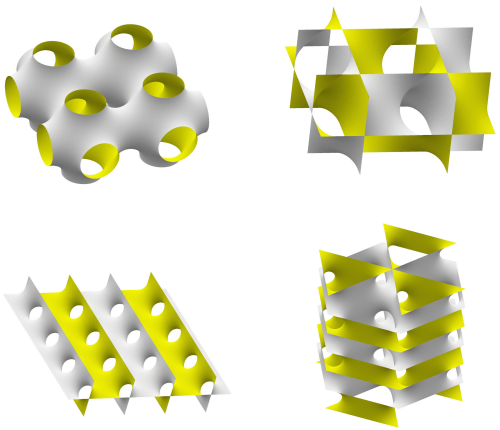
Mechanical equilibrium

A particle is in mechanical equilibrium if the net force on that particle is zero. By extension, a **physical system** made up of many parts is in mechanical equilibrium if the **net force on each of its individual parts is zero**. In addition to defining mechanical equilibrium in terms of force, there are many alternative definitions for mechanical equilibrium which are all mathematically equivalent. In terms of momentum, a system is in equilibrium if the momentum of its parts is all constant. In terms of velocity, the **system is in equilibrium if velocity is constant**. Since all particles in equilibrium have constant velocity, it is always possible to find an inertial reference frame in which the **particle is stationary** with respect to the frame.

Minimal surface

A surface that locally **minimizes its area**. This is equivalent to having **zero mean curvature**.
The term minimal surface is used because these surfaces originally arose as surfaces that minimized total surface area **subject to some constraint**. Physical models of area-minimizing minimal surfaces can be made by dipping a wire frame into a **soap solution**, forming a soap film, which is a minimal surface whose boundary is the wire frame. For a given constraint there may also exist several minimal surfaces with different areas.

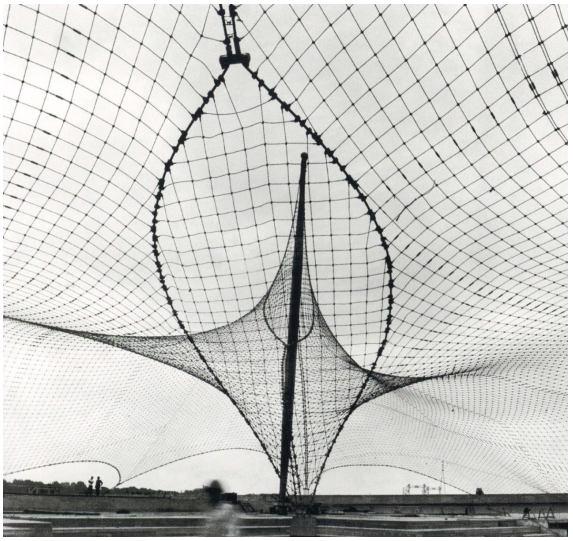
*A **gyroid** is an infinitely connected triply periodic minimal surface discovered by Alan Schoen in 1970*



*Examples of **Schwarz minimal surfaces**: primitive (upper left), diamond (upper right), crossed layers of parallels (lower left), hexagonal (lower right)*

Stretched grid method

A numerical technique for finding approximate solutions of various mathematical and engineering problems that can be related to an **elastic grid behavior**. In particular, meteorologists use the stretched grid method for weather prediction and engineers use the stretched grid method to design tents and other **tensile structures**.

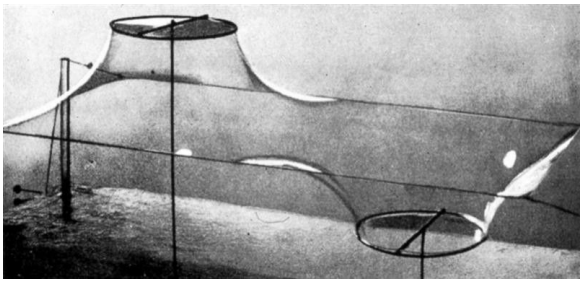


West German Pavilion at Expo 67 in Montreal by Frei Otto is an example of a stretched grid structure

Soap films

Physical examples of the complex mathematical problem of **minimal surface**. They will assume the shape of least surface area possible containing a given volume. A soap film, which has equal pressure on inside as outside, is a surface with zero mean curvature. A soap bubble is a closed soap film: due to the difference in outside and inside pressure, it is a surface of constant mean curvature. While it has been known since 1884 that a spherical soap bubble is the **least-area way of enclosing** a given volume of air (a theorem of H. A. Schwarz), it was not until 2000 that it was proven that two merged soap bubbles provide the optimum way of enclosing two given volumes of air of different size with the least surface area. This has been dubbed the **double bubble conjecture**.

Because of these qualities, soap bubble films have been used with practical problem solving application. Structural engineer **Frei Otto** used soap bubble films to determine the geometry of a sheet of least surface area that spreads between several points, and translated this geometry into revolutionary tensile roof structures.

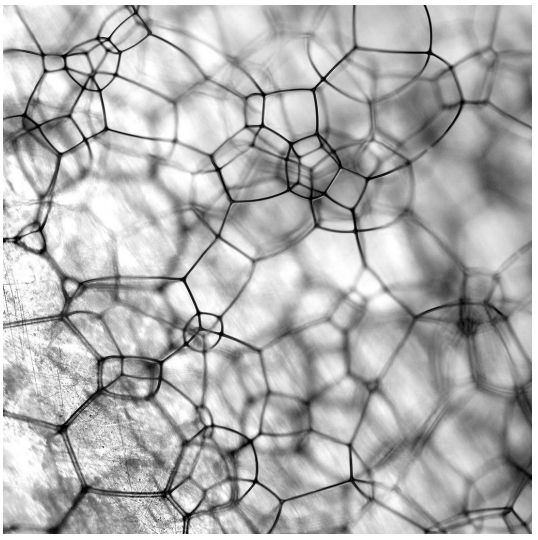


Soap film experiments conducted at Institute for Lightweight Structures at the University of Stuttgart, which Frei Otto founded in 1964

The structures that soap films make can not just be enclosed as spheres, but virtually any shape, for example in wire frames. Therefore, many different minimal surfaces can be designed. It is actually sometimes easier to physically make them than to compute them by mathematical modelling. This is why the soap films can be considered as **analog computers** which can in some cases outperform conventional computers, depending on the complexity of the system.

Weaire–Phelan structure

A complex 3-dimensional structure representing an **idealised foam of equal-sized bubbles**. It uses two kinds of cells which have equal volume. The first one is a **pyritohedron**, an irregular dodecahedron with pentagonal faces. The second is a form of **truncated hexagonal trapezohedron**, a species of tetrakaidecahedron with two hexagonal and twelve pentagonal faces. The Weaire–Phelan structure is the inspiration for the design of the **Beijing National Aquatics Centre** for the 2008 Olympics in Beijing in China.



Soap bubbles forming a Weaire–Phelan structure as it encloses a volume with a least surface area

In 1993, Trinity College Dublin physicist **Denis Weaire** and his student **Robert Phelan** found that in computer simulations of foam, this structure was a better solution of the **Kelvin problem** than the previous best-known solution, the Kelvin structure.

In 1887, Lord Kelvin asked how space could be partitioned into **cells of equal volume** with the **least area of surface** between them, i.e., what was the most efficient bubble foam? This problem has since been referred to as the **Kelvin problem**. He proposed a foam, based on the bitruncated cubic honeycomb, which is called the **Kelvin structure**. This is the convex uniform honeycomb formed by the truncated octahedron, which is a 14-faced space-filling polyhedron (a tetradecahedron), with 6 square faces and 8 hexagonal faces. The **Kelvin conjecture** is that this structure solves the Kelvin problem: that the foam of the bitruncated cubic honeycomb is the most efficient foam. The Kelvin conjecture was widely believed, and no counterexample was known for more than 100 years, until it was **disproved by the discovery of the Weaire–Phelan structure**.