KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

„Image Processing and Computer Graphics”

Prezentacja multimedialna współfinansowana przez Unię Europejską w ramach Europejskiego Funduszu Społecznego w projekcie pt.

„Innowacyjna dydaktyka bez ograniczeń - zintegrowany rozwój Politechniki Łódzkiej - zarządzanie Uczelnią, nowoczesna oferta edukacyjna i wzmacniania zdolności do zatrudniania osób niepełnosprawnych”

Politechnika Łódzka, ul. Żeromskiego 116, 90-924 Łódź, tel. (042) 631 28 83
www.kapitalludzki.p.lodz.pl

Contribution

Lecturer:

Piotr M. Szczypiński, Dr inż.
Institute of Electronics, Technical University of Lodz
Wolczanska 211/215, 90-924 Lodz, Poland
tel. +48426312638
http://www.eletel.p.lodz.pl/pms
pms@p.lodz.pl

Fragments of the lecture were also provided by

Andrzej Materka, Michał Strzelecki, Paweł Strumiłło, Piotr Romaniuk, Piotr Makowski

and were used with permission.

Portions of this lecture are illustrated with articles from the wikipedia
(GNU Free Documentation License
http://en.wikipedia.org/wiki/Wikipedia:Text_of_the_GNU_Free_Documentation_License)

3D vector graphics

- Co-ordinate systems
- Projection and transformation (linear algebra)
- Light and shading
- Rendering methods


3D modeling
From Wikipedia, the free encyclopedia
This article does not cite any references or sources. Please help improve this article by adding citations to reliable sources (ideally, using inline citations). Unsourced material may be challenged and removed. (April 2007)
This article is about computer modeling within an artistic medium. For scientific usage, see Computer simulation.
In 3D computer graphics, 3D modeling is the process of developing a mathematical, wireframe representation of any three-dimensional object (either inanimate or living) via specialized software. The product is called a 3D model. It can be displayed as a two-dimensional image through a process called 3D rendering or used in a computer simulation of physical phenomena. The model can also be physically created using 3D Printing devices.
Models may be created automatically or manually. The manual modeling process of preparing geometric data for 3D computer graphics is similar to plastic arts such as sculpting.
Contents [hide]
1 Models
1.1 Representation
2 Modeling processes
3 Scene setup
4 Compared to 2D methods
5 3D model market
6 See also
7 References
8 External links
Models [edit]
3D computer graphics
Basics
3D modeling / 3D scanning
3D rendering / 3D printing
3D computer graphics software
Primary Uses
3D models / Computer-aided design
Graphic design / Video games
Visual effects / Visualization
Virtual engineering / Virtual reality

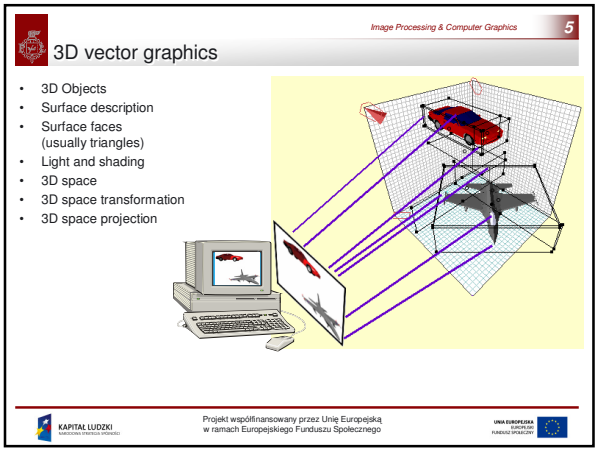


Image Processing & Computer Graphics
5
3D vector graphics
3D Objects
Surface description
Surface faces (usually triangles)
Light and shading
3D space
3D space transformation
3D space projection
KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

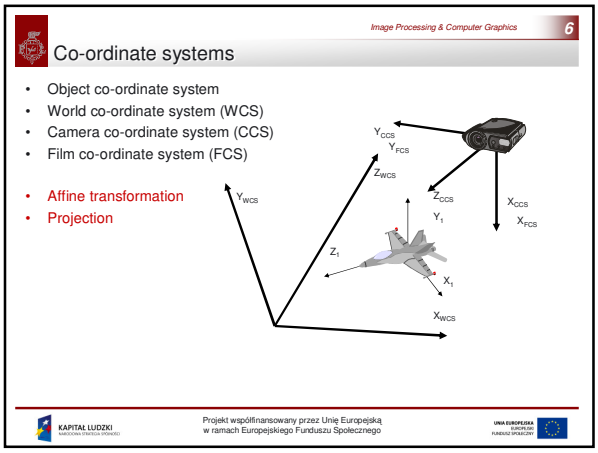
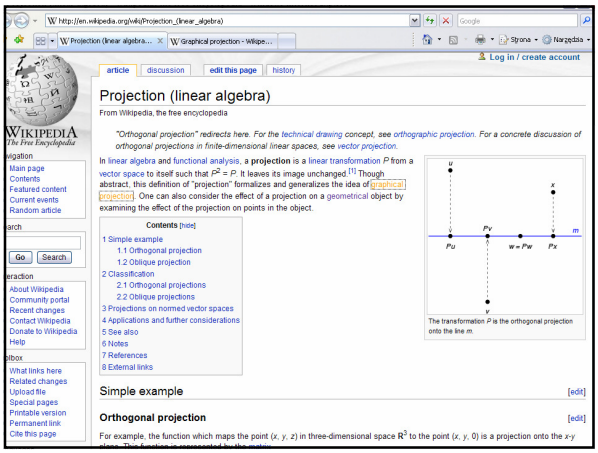


Image Processing & Computer Graphics
6
Co-ordinate systems
Object co-ordinate system
World co-ordinate system (WCS)
Camera co-ordinate system (CCS)
Film co-ordinate system (FCS)
Affine transformation
Projection
KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY



Projection (linear algebra)

From Wikipedia, the free encyclopedia

"Orthogonal projection" redirects here. For the technical drawing concept, see orthographic projection. For a concrete discussion of orthogonal projections in finite-dimensional linear spaces, see vector projection.

In linear algebra and functional analysis, a **projection** is a linear transformation P from a vector space to itself such that $P^2 = P$. It leaves its image unchanged.[1] Though abstract, this definition of "projection" formalizes and generalizes the idea of graphical projection. One can also consider the effect of a projection on a geometrical object by examining the effect of the projection on points in the object.

Contents [hide]

1 Simple example
- 1.1 Orthogonal projection
- 1.2 Oblique projection

2 Classification
- 2.1 Orthogonal projections
- 2.2 Oblique projections

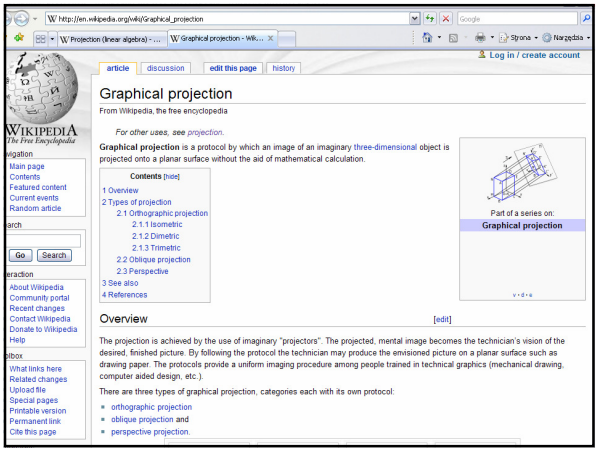
3 Projections on normed vector spaces
4 Applications and further considerations
5 See also
6 Notes
7 References
8 External links

The transformation P is the orthogonal projection onto the line m.

Simple example [edit]

Orthogonal projection [edit]

For example, the function which maps the point (x, y, z) in three-dimensional space $\mathbf{R}^3$ to the point $(x, y, 0)$ is a projection onto the x-y plane.



Graphical projection

From Wikipedia, the free encyclopedia

For other uses, see projection.

Graphical projection is a protocol by which an image of an imaginary three-dimensional object is projected onto a planar surface without the aid of mathematical calculation.

Contents [hide]

1 Overview
2 Types of projection
- 2.1 Orthographic projection
  - 2.1.1 Isometric
  - 2.1.2 Dimetric
  - 2.1.3 Trimetric
- 2.2 Oblique projection
- 2.3 Perspective

3 See also
4 References

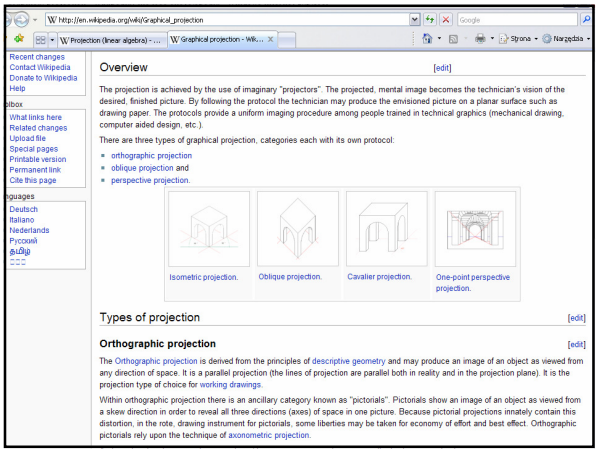
Part of a series on:
Graphical projection

Overview [edit]

The projection is achieved by the use of imaginary "projectors". The projected, mental image becomes the technician's vision of the desired, finished picture. By following the protocol the technician may produce the envisioned picture on a planar surface such as drawing paper. The protocols provide a uniform imaging procedure among people trained in technical graphics (mechanical drawing, computer aided design, etc.).

There are three types of graphical projection, categories each with its own protocol:

- orthographic projection
- oblique projection and
- perspective projection.



Overview [edit]

The projection is achieved by the use of imaginary "projectors". The projected, mental image becomes the technician's vision of the desired, finished picture. By following the protocol the technician may produce the envisioned picture on a planar surface such as drawing paper. The protocols provide a uniform imaging procedure among people trained in technical graphics (mechanical drawing, computer aided design, etc.).

There are three types of graphical projection, categories each with its own protocol:

- orthographic projection
- oblique projection and
- perspective projection.

Isometric projection. | Oblique projection. | Cavalier projection. | One-point perspective projection.

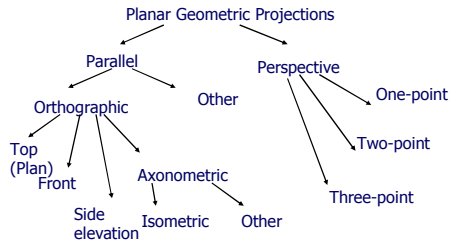
Types of projection [edit]

Orthographic projection [edit]

The Orthographic projection is derived from the principles of descriptive geometry and may produce an image of an object as viewed from any direction of space. It is a parallel projection (the lines of projection are parallel both in reality and in the projection plane). It is the projection type of choice for working drawings.

Within orthographic projection there is an ancillary category known as "pictorials". Pictorials show an image of an object as viewed from a skew direction in order to reveal all three directions (axes) of space in one picture. Because pictorial projections innately contain this distortion, in the rote, drawing instrument for pictorials, some liberties may be taken for economy of effort and best effect. Orthographic pictorials rely upon the technique of axonometric projection.

Projection Classification

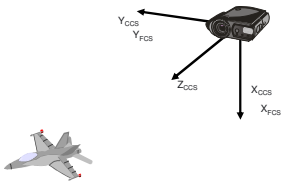


„Introduction to Computer Graphics"
J.Foley, A. Van Dam, S. Feiner, R. Phillips
Addison-Wesley Pub. Comp. 1995

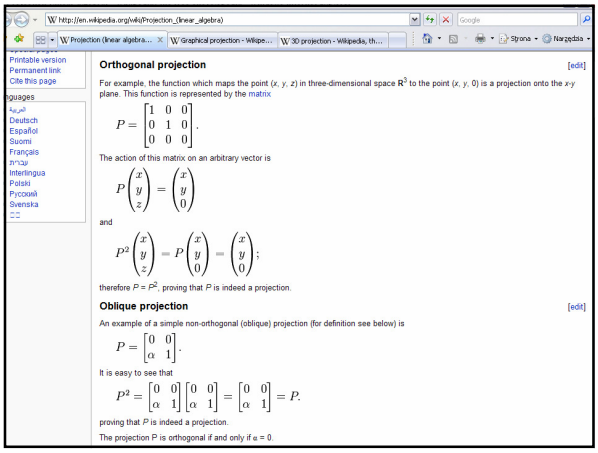
KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Projection

- Co-ordinates of objects are given in CCS (already transformed).
- Projection is to reduce dimensionality of 3D space to 2D space (of a film)



KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY



Printable version
Permanent link
Cite this page

Languages
العربية
Deutsch
Español
Suomi
Français
עברית
Interlingua
Polski
Русский
Svenska

Orthogonal projection [edit]

For example, the function which maps the point (x, y, z) in three-dimensional space $\mathbf{R}^3$ to the point $(x, y, 0)$ is a projection onto the x-y plane. This function is represented by the matrix

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The action of this matrix on an arbitrary vector is

$$P \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}$$

and

$$P^2 \begin{pmatrix} x \\ y \\ z \end{pmatrix} = P \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix};$$

therefore $P = P^2$, proving that P is indeed a projection.

Oblique projection [edit]

An example of a simple non-orthogonal (oblique) projection (for definition see below) is

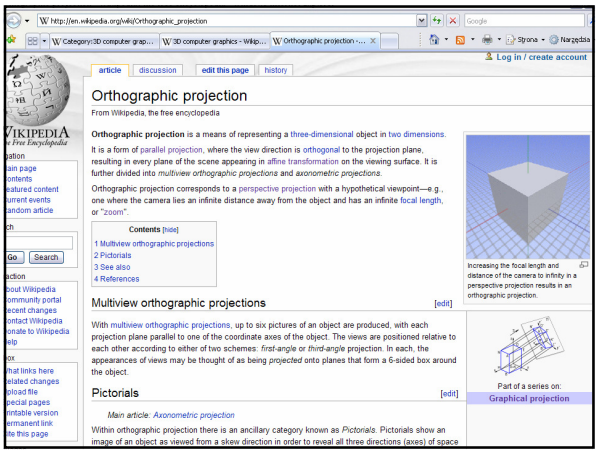
$$P = \begin{bmatrix} 0 & 0 \\ \alpha & 1 \end{bmatrix}.$$

It is easy to see that

$$P^2 = \begin{bmatrix} 0 & 0 \\ \alpha & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ \alpha & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \alpha & 1 \end{bmatrix} = P.$$

proving that P is indeed a projection.

The projection P is orthogonal if and only if $\alpha = 0$.



Orthographic projection

From Wikipedia, the free encyclopedia

Orthographic projection is a means of representing a three-dimensional object in two dimensions.

It is a form of parallel projection, where the view direction is orthogonal to the projection plane, resulting in every plane of the scene appearing in affine transformation on the viewing surface. It is further divided into *multiview orthographic projections* and *axonometric projections*.

Orthographic projection corresponds to a perspective projection with a hypothetical viewpoint—e.g., one where the camera lies an infinite distance away from the object and has an infinite focal length, or "zoom".

Contents [hide]

1 Multiview orthographic projections
2 Pictorials
3 See also
4 References

Increasing the focal length and distance of the camera to infinity in a perspective projection results in an orthographic projection.

Multiview orthographic projections [edit]

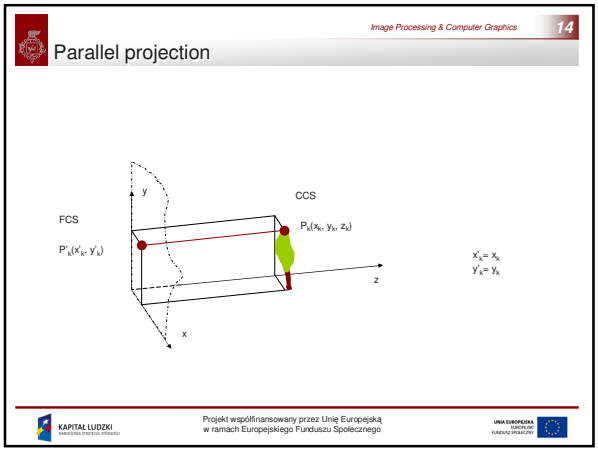
With multiview orthographic projections, up to six pictures of an object are produced, with each projection plane parallel to one of the coordinate axes of the object. The views are positioned relative to each other according to either of two schemes: *first-angle* or *third-angle* projection. In each, the appearances of views may be thought of as being *projected* onto planes that form a 6-sided box around the object.

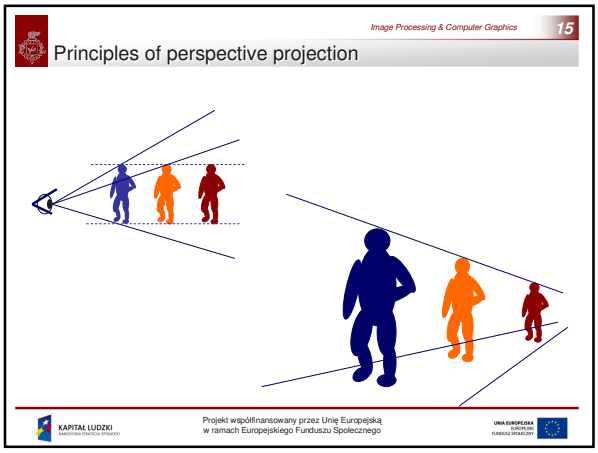
Pictorials [edit]

Main article: Axonometric projection

Within orthographic projection there is an ancillary category known as *Pictorials*. Pictorials show an image of an object as viewed from a skew direction in order to reveal all three directions (axes) of space

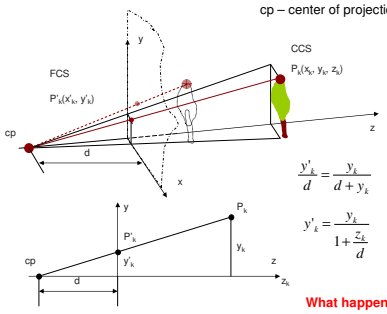
Part of a series on:
Graphical projection





Perspective projection

cp – center of projection



$$\frac{y'_k}{d} = \frac{y_k}{d + y_k} \qquad \frac{x'_k}{d} = \frac{x_k}{d + z_k}$$

$$y'_k = \frac{y_k}{1 + \frac{z_k}{d}} \qquad x'_k = \frac{x_k}{1 + \frac{z_k}{d}}$$

What happens when *d* is huge?

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

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

Affine transformation - W... | Linear map - Wikipedia, the f...

Log in / create account

article | discussion | edit this page | history

Affine transformation

From Wikipedia, the free encyclopedia

This article includes a list of references or external links, but its sources remain unclear because it lacks inline citations. Please improve this article by introducing more precise citations where appropriate. (March 2009)

In geometry, an **affine transformation** or **affine map** or an **affinity** (from the Latin, *affinis*, "connected with") between two vector spaces (strictly speaking, two affine spaces) consists of a linear transformation followed by a translation:

$$x \mapsto Ax + b.$$

In the finite-dimensional case each affine transformation is given by a matrix A and a vector *b*, satisfying certain properties described below.

Geometrically, an affine transformation in Euclidean space is one that preserves

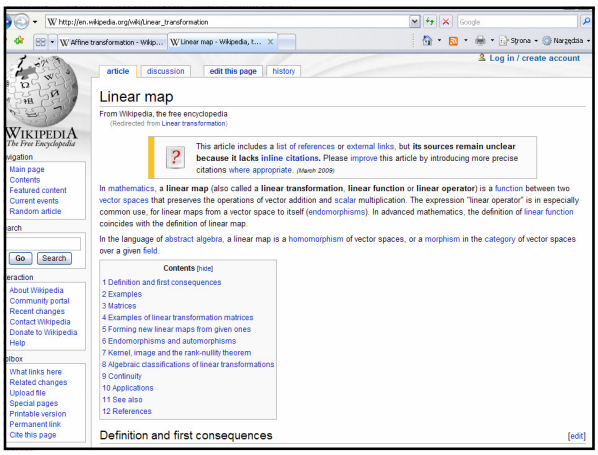
1. The collinearity relation between points; i.e., three points which lie on a line continue to be collinear after the transformation
2. Ratios of distances along a line; i.e., for distinct collinear points p_1, p_2, p_3, the ratio $|p_2 - p_1| / |p_3 - p_2|$ is preserved

In general, an affine transform is composed of linear transformations (rotation, scaling or shear) and a translation (or "shift"). Several linear transformations can be combined into a single one, so that the general formula given above is still applicable.

Contents [hide]

1 Representation of affine transformations
2 Properties of affine transformations
3 Affine transformations and linear transformations
4 Affine transformation of the plane
5 Example of an affine transformation
6 See also
7 External links

Representation of affine transformations [edit]



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

Affine transformation - Wikip... | Linear map - Wikipedia, t...

Log in / create account

article | discussion | edit this page | history

Linear map

From Wikipedia, the free encyclopedia
(Redirected from Linear transformation)

This article includes a list of references or external links, but its sources remain unclear because it lacks inline citations. Please improve this article by introducing more precise citations where appropriate. (March 2009)

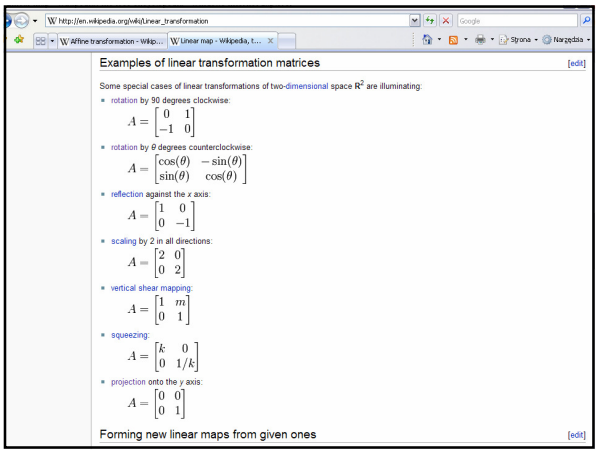
In mathematics, a **linear map** (also called a **linear transformation**, **linear function** or **linear operator**) is a function between two vector spaces that preserves the operations of vector addition and scalar multiplication. The expression "linear operator" is in especially common use, for linear maps from a vector space to itself (endomorphisms). In advanced mathematics, the definition of linear function coincides with the definition of linear map.

In the language of abstract algebra, a linear map is a homomorphism of vector spaces, or a morphism in the category of vector spaces over a given field.

Contents [hide]

1 Definition and first consequences
2 Examples
3 Matrices
4 Examples of linear transformation matrices
5 Forming new linear maps from given ones
6 Endomorphisms and automorphisms
7 Kernel, image and the rank-nullity theorem
8 Algebraic classifications of linear transformations
9 Continuity
10 Applications
11 See also
12 References

Definition and first consequences [edit]

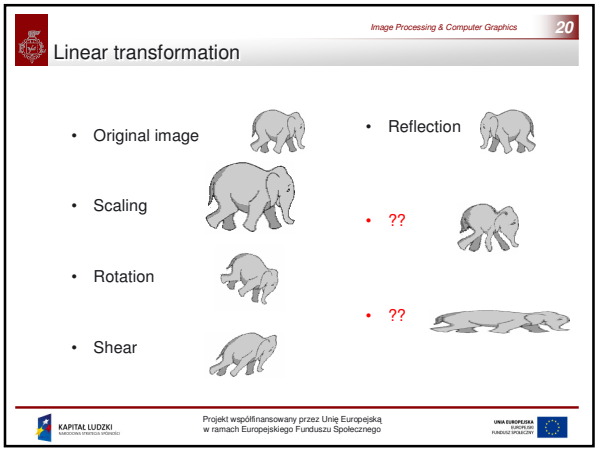


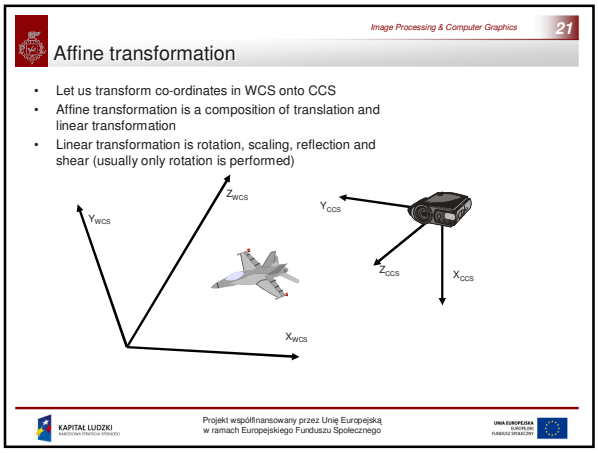
Examples of linear transformation matrices [edit]

Some special cases of linear transformations of two-dimensional space R^2 are illuminating:

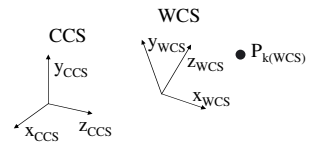
- rotation by 90 degrees clockwise:
$$A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$
- rotation by θ degrees counterclockwise:
$$A = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$
- reflection against the x axis:
$$A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$
- scaling by 2 in all directions:
$$A = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$
- vertical shear mapping:
$$A = \begin{bmatrix} 1 & m \\ 0 & 1 \end{bmatrix}$$
- squeezing:
$$A = \begin{bmatrix} k & 0 \\ 0 & 1/k \end{bmatrix}$$
- projection onto the y axis:
$$A = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Forming new linear maps from given ones [edit]

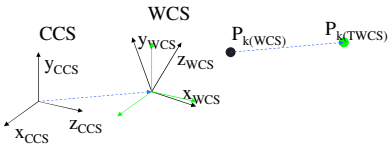




Translation

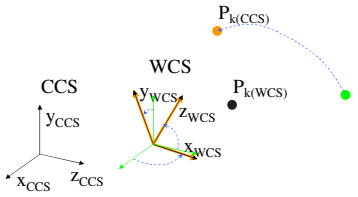


$$\begin{bmatrix} x_{k(TWCS)} \\ y_{k(TWCS)} \\ z_{k(TWCS)} \end{bmatrix} = \begin{bmatrix} x_{k(WCS)} \\ y_{k(WCS)} \\ z_{k(WCS)} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$



Linear transformation (Rotation matrix)

$$\begin{bmatrix} x_{k(CCS)} \\ y_{k(CCS)} \\ z_{k(CCS)} \end{bmatrix} = \begin{bmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{bmatrix} \left(\begin{bmatrix} x_{k(WCS)} \\ y_{k(WCS)} \\ z_{k(WCS)} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \right)$$



Rotation matrix properties

- Rotation means no scaling, shear or reflection
- Properties:
  - Orthonormal
    - |det **C**| = 1 – no scaling
    - det **C** > 0 – no reflection
    - Orthogonal – no shear

$$\begin{bmatrix} x_{k(CCS)} \\ y_{k(CCS)} \\ z_{k(CCS)} \end{bmatrix} = \underbrace{\begin{bmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{bmatrix}}_{\mathbf{C}} \left(\begin{bmatrix} x_{k(WCS)} \\ y_{k(WCS)} \\ z_{k(WCS)} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \right)$$

$$\mathbf{C}^T\mathbf{C}=1$$

$$\mathbf{C}^T\mathbf{C}\mathbf{C}^{-1}=1\mathbf{C}^{-1}$$

$$\mathbf{C}^T=\mathbf{C}^{-1}$$

Example 1

1. Camera is at $[a_x, a_y, a_z]^T$ in WCS,
2. Camera is directed along $[b_x, b_y, b_z]^T$ vector in WCS,
3. Camera stands straight (base is parallel to OYZ plane of WCS)

Find translation vector and rotation matrix

Example 1

1. Camera is at $[a_x, a_y, a_z]^T$ in WCS. Therefore translation vector $[t_x, t_y, t_z]^T = -[a_x, a_y, a_z]^T$
2. Vector $[c_{zx}, c_{zy}, c_{zz}]^T$ transforms camera direction in WCS into Z-axis in CCS, Therefore it is equal to normalized cmera direction
   $[c_{zx}, c_{zy}, c_{zz}]^T = [b_x, b_y, b_z]^T / ||[b_x, b_y, b_z]^T||$
3. Camera stands straight. Vector $[c_{zx}, c_{zy}, c_{zz}]^T$ is perpendicular $[c_{zx}, c_{zy}, c_{zz}]^T$ and parallel to Y axis and Z axis of WCS.
   $[c_{xx}, c_{xy}, c_{xz}]^T = [c_{zz}, 0, -c_{zx}]^T / ||[c_{zz}, 0, -c_{zx}]^T||$
4. Vector $[c_{yx}, c_{yy}, c_{yz}]^T$ is parallel to $[c_{xx}, c_{xy}, c_{xz}]^T$ and $[c_{zx}, c_{zy}, c_{zz}]^T$. Therefore it is a vector product of the two.
   $[c_{yx}, c_{yy}, c_{yz}]^T = [c_{xx}, c_{xy}, c_{xz}]^T \times [c_{zx}, c_{zy}, c_{zz}]^T$

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

$$\begin{bmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{bmatrix}$$

Example 2

1. Camera is at $[a_x, a_y, a_z]^T$ in WCS,
2. Camera is rotated by:
   yaw α angle to the left
   pitch β angle up
   and rolled by γ angle counter clockwise

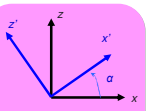
Find translation vector and rotation matrix

Example 2

1. $[t_x, t_y, t_z]^T = -[a_x, a_y, a_z]^T$
2. Rotation matrix is composed by three rotations

yaw, pitch and roll.

The three matrices are multiplied to produce a complete rotation in 3D space.



$x' = x\cos\alpha + z\sin\alpha$
$z' = z\cos\alpha - x\sin\alpha$

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

Example 2

$$\begin{bmatrix} x_{k(CCS)} \\ y_{k(CCS)} \\ z_{k(CCS)} \end{bmatrix} = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \left(\begin{bmatrix} x_{k(WCS)} \\ y_{k(WCS)} \\ z_{k(WCS)} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \right)$$

Homogeneous coordinates

Affine transformation is a composition of translation and linear transformation (summation of vectors and matrix multiplication).

$$\begin{bmatrix} x_{k(CCS)} \\ y_{k(CCS)} \\ z_{k(CCS)} \end{bmatrix} = \begin{bmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{bmatrix} \left(\begin{bmatrix} x_{k(WCS)} \\ y_{k(WCS)} \\ z_{k(WCS)} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \right)$$

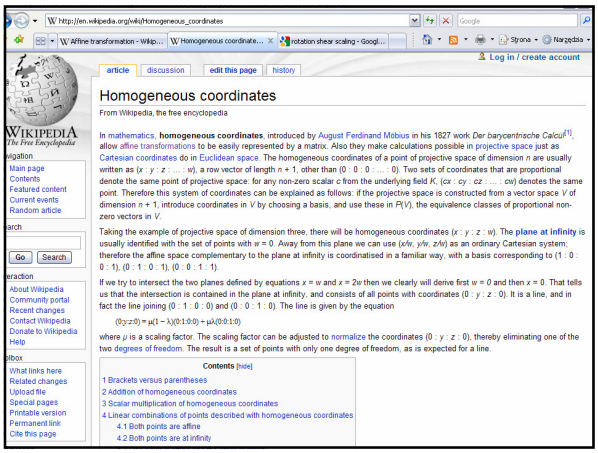
The affine transformation is represented by transation vector and rotation matrix.

However, the affine transformation can be represented just by a single matrix in homogeneous coordinates.

$$\begin{bmatrix} x_{k(CCS)} \\ y_{k(CCS)} \\ z_{k(CCS)} \\ 1 \end{bmatrix} = \begin{bmatrix} c_{xx} & c_{xy} & c_{xz} & 0 \\ c_{yx} & c_{yy} & c_{yz} & 0 \\ c_{zx} & c_{zy} & c_{zz} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k(WCS)} \\ y_{k(WCS)} \\ z_{k(WCS)} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_{k(CCS)} \\ y_{k(CCS)} \\ z_{k(CCS)} \\ 1 \end{bmatrix} = \begin{bmatrix} c_{xx} & c_{xy} & c_{xz} & c_{xx}t_x + c_{xy}t_y + c_{xz}t_z \\ c_{yx} & c_{yy} & c_{yz} & c_{yx}t_x + c_{yy}t_y + c_{yz}t_z \\ c_{zx} & c_{zy} & c_{zz} & c_{zx}t_x + c_{zy}t_y + c_{zz}t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k(WCS)} \\ y_{k(WCS)} \\ z_{k(WCS)} \\ 1 \end{bmatrix}$$

A 4x4 transformation matrix is often used in 3D graphics programs.



Homogeneous coordinates

From Wikipedia, the free encyclopedia

In mathematics, **homogeneous coordinates**, introduced by August Ferdinand Möbius in his 1827 work *Der barycentrische Calcül*[1], allow affine transformations to be easily represented by a matrix. Also they make calculations possible in projective space just as Cartesian coordinates do in Euclidean space. The homogeneous coordinates of a point of projective space of dimension n are usually written as $(x : y : z : \ldots : w)$, a row vector of length $n + 1$, other than $(0 : 0 : 0 : \ldots : 0)$. Two sets of coordinates that are proportional denote the same point of projective space: for any non-zero scalar c from the underlying field K, $(cx : cy : cz : \ldots : cw)$ denotes the same point. Therefore this system of coordinates can be explained as follows: if the projective space is constructed from a vector space V of dimension $n + 1$, introduce coordinates in V by choosing a basis, and use these in $P(V)$, the equivalence classes of proportional non-zero vectors in V.

Talking the example of projective space of dimension three, there will be homogeneous coordinates $(x : y : z : w)$. The **plane at infinity** is usually identified with the set of points with $w = 0$. Away from this plane we can use $(x/w, y/w, z/w)$ as an ordinary Cartesian system; therefore the affine space complementary to the plane at infinity is coordinatised in a familiar way, with a basis corresponding to $(1 : 0 : 0 : 1)$, $(0 : 1 : 0 : 1)$, $(0 : 0 : 1 : 1)$.

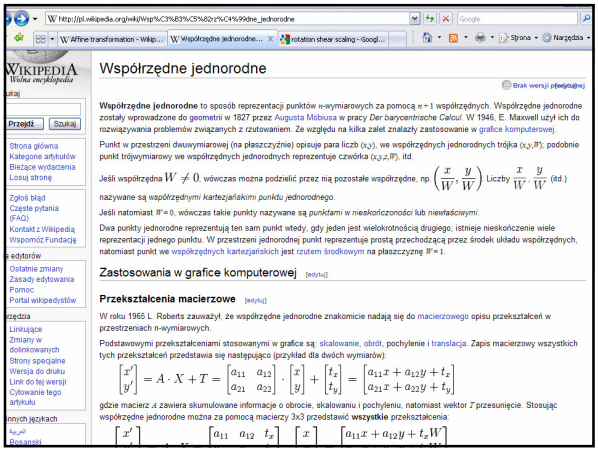
If we try to intersect the two planes defined by equations $x = w$ and $x = 2w$ then we clearly will derive first $w = 0$ and then $x = 0$. That tells us that the intersection is contained in the plane at infinity, and consists of all points with coordinates $(0 : y : z : 0)$. It is a line, and in fact the line joining $(0 : 1 : 0 : 0)$ and $(0 : 0 : 1 : 0)$. The line is given by the equation

$$(0:y:z:0) = \mu(1-\lambda)(0:1:0:0) + \mu\lambda(0:0:1:0)$$

where μ is a scaling factor. The scaling factor can be adjusted to normalize the coordinates $(0 : y : z : 0)$, thereby eliminating one of the two degrees of freedom. The result is a set of points with only one degree of freedom, as is expected for a line.

Contents [hide]

1 Brackets versus parentheses
2 Addition of homogeneous coordinates
3 Scalar multiplication of homogeneous coordinates
4 Linear combinations of points described with homogeneous coordinates
4.1 Both points are affine
4.2 Both points are at infinity



Współrzędne jednorodne

Współrzędne jednorodne to sposób reprezentacji punktów n-wymiarowych za pomocą $n+1$ współrzędnych. Współrzędne jednorodne zostały wprowadzone do geometrii w 1827 przez Augusta Möbiusa w pracy *Der barycentrische Calcul*. W 1946, E. Maxwell użył ich do rozwiązywania problemów związanych z rzutowaniem. Ze względu na kilka zalet znalazły zastosowanie w grafice komputerowej.

Punkt w przestrzeni dwuwymiarowej (na płaszczyźnie) opisuje para liczb (x,y), we współrzędnych jednorodnych trójka (x,y,W); podobnie punkt trójwymiarowy we współrzędnych jednorodnych reprezentuje czwórka (x,y,z,W), itd.

Jeśli współrzędna $W \neq 0$, wówczas można podzielić przez nią pozostałe współrzędne, np. $\left(\frac{x}{W}, \frac{y}{W}\right)$ Liczby $\frac{x}{W}, \frac{y}{W}$ (itd.) nazywane są *współrzędnymi kartezjańskimi punktu jednorodnego*.

Jeśli natomiast $W = 0$, wówczas takie punkty nazywane są *punktami w nieskończoności* lub *niewłaściwymi*.

Dwa punkty jednorodne reprezentują ten sam punkt wtedy, gdy jeden jest wielokrotnością drugiego; istnieje nieskończenie wiele reprezentacji jednego punktu. W przestrzeni jednorodnej punkt reprezentuje prostą przechodzącą przez środek układu współrzędnych, natomiast punkt we współrzędnych kartezjańskich jest rzutem środkowym na płaszczyznę $W = 1$.

Zastosowania w grafice komputerowej

Przekształcenia macierzowe

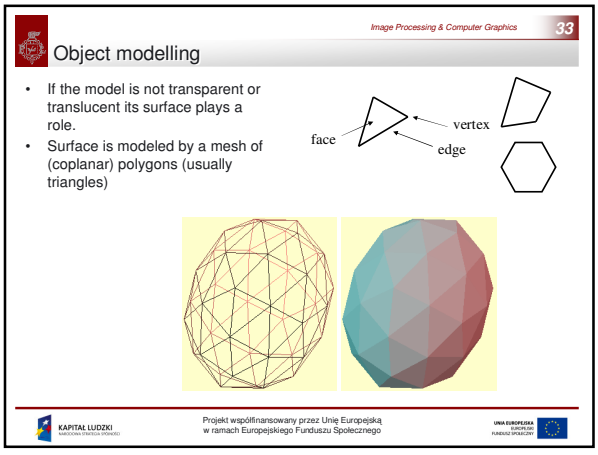
W roku 1965 L. Roberts zauważył, że współrzędne jednorodne znakomicie nadają się do macierzowego opisu przekształceń w przestrzeniach n-wymiarowych.

Podstawowymi przekształceniami stosowanymi w grafice są: skalowanie, obrót, pochylenie i translacja. Zapis macierzowy wszystkich tych przekształceń przedstawia się następująco (przykład dla dwóch wymiarów):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = A \cdot X + T = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + t_x \\ a_{21}x + a_{22}y + t_y \end{bmatrix}$$

gdzie macierz A zawiera skumulowane informacje o obrocie, skalowaniu i pochyleniu, natomiast wektor T przesunięcie. Stosując współrzędne jednorodne można za pomocą macierzy 3x3 przedstawić **wszystkie** przekształcenia:

$$\begin{bmatrix} x' \\ \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \end{bmatrix} \begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + t_xW \end{bmatrix}$$

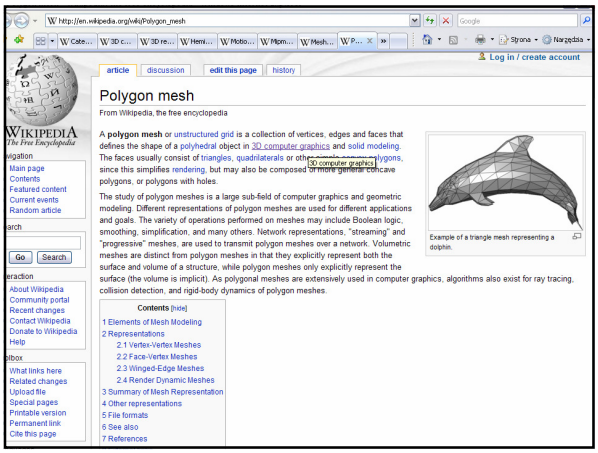


Object modelling

- If the model is not transparent or translucent its surface plays a role.
- Surface is modeled by a mesh of (coplanar) polygons (usually triangles)

face
vertex
edge

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY



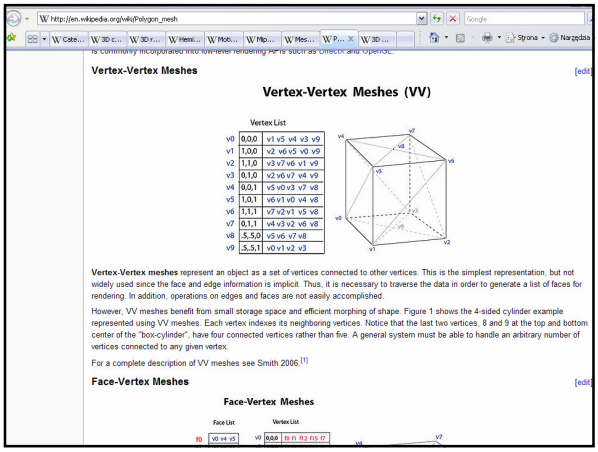
Polygon mesh

From Wikipedia, the free encyclopedia

A **polygon mesh** or unstructured grid is a collection of vertices, edges and faces that defines the shape of a polyhedral object in 3D computer graphics and solid modeling. The faces usually consist of triangles, quadrilaterals or other simple convex polygons; since this simplifies rendering, but may also be composed of more general concave polygons, or polygons with holes.

The study of polygon meshes is a large sub-field of computer graphics and geometric modeling. Different representations of polygon meshes are used for different applications and goals. The variety of operations performed on meshes may include Boolean logic, smoothing, simplification, and many others. Network representations, "streaming" and "progressive" meshes, are used to transmit polygon meshes over a network. Volumetric meshes are distinct from polygon meshes in that they explicitly represent both the surface and volume of a structure, while polygon meshes only explicitly represent the surface (the volume is implicit). As polygonal meshes are extensively used in computer graphics, algorithms also exist for ray tracing, collision detection, and rigid-body dynamics of polygon meshes.

Example of a triangle mesh representing a dolphin.



Vertex-Vertex Meshes

Vertex-Vertex Meshes (VV)

Vertex List

v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,0	v5 v6 v7 v8
v9	.5,.5,1	v0 v1 v2 v3

Vertex-Vertex meshes represent an object as a set of vertices connected to other vertices. This is the simplest representation, but not widely used since the face and edge information is implicit. Thus, it is necessary to traverse the data in order to generate a list of faces for rendering. In addition, operations on edges and faces are not easily accomplished.

However, VV meshes benefit from small storage space and efficient morphing of shape. Figure 1 shows the 4-sided cylinder example represented using VV meshes. Each vertex indexes its neighboring vertices. Notice that the last two vertices, 8 and 9 at the top and bottom center of the "box-cylinder", have four connected vertices rather than five. A general system must be able to handle an arbitrary number of vertices connected to any given vertex.

For a complete description of VV meshes see Smith 2006.[1]

Face-Vertex Meshes

Face-Vertex Meshes

Face List

Vertex List

Excluding (culling) hidden surfaces

- Viewing frustum culling (clipping)
- Back face culling
- Occlusion culling

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Hidden surface determination

From Wikipedia, the free encyclopedia

In 3D computer graphics, **hidden surface determination** (also known as **Hidden Surface Removal** (HSR), or **Visible Surface Determination** (VSD).) is the process used to determine which surfaces and parts of surfaces are not visible from a certain viewpoint. A hidden surface determination algorithm is a solution to the visibility problem, which was one of the first major problems in the field of 3D computer graphics. The process of hidden surface determination is sometimes called **hiding**, and such an algorithm is sometimes called a **hider**. The analogue for line rendering is hidden line removal. Hidden surface determination is necessary to render an image correctly, so that one cannot look through walls in virtual reality, for example.

There are many techniques for hidden surface determination. They are fundamentally an exercise in sorting, and usually vary in the order in which the sort is performed and how the problem is subdivided. Sorting large quantities of graphics primitives is usually done by divide and conquer.

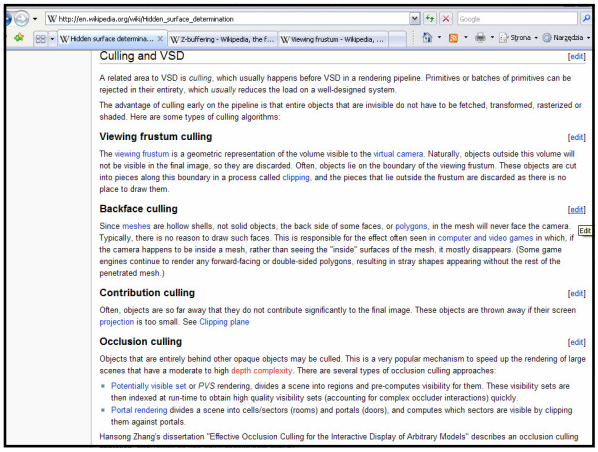
Contents [hide]

1 Hidden surface removal algorithms
2 Culling and VSD
2.1 Viewing frustum culling
2.2 Backface culling
2.3 Contribution culling
2.4 Occlusion culling
3 Divide and conquer

Hidden surface removal algorithms

Considering the rendering pipeline, the projection, the clipping, and the rasterization steps are handled differently by the following algorithms:

- Z-buffering During rasterization the depth/Z value of each pixel (or *sample* in the case of anti-aliasing, but without loss of generality we use the term *pixel*) is checked against an existing depth value. If the current pixel is behind the pixel in the Z-buffer, the pixel is rejected, otherwise it is shaded and its depth value replaces the one in the Z-buffer. Z-buffering supports dynamic scenes easily, and is currently implemented efficiently in graphics hardware. This is the current standard. The cost of using Z-buffering is that it uses up



Culling and VSD

A related area to VSD is *culling*, which usually happens before VSD in a rendering pipeline. Primitives or batches of primitives can be rejected in their entirety, which *usually* reduces the load on a well-designed system.

The advantage of culling early on the pipeline is that entire objects that are invisible do not have to be fetched, transformed, rasterized or shaded. Here are some types of culling algorithms:

Viewing frustum culling

The viewing frustum is a geometric representation of the volume visible to the virtual camera. Naturally, objects outside this volume will not be visible in the final image, so they are discarded. Often, objects lie on the boundary of the viewing frustum. These objects are cut into pieces along this boundary in a process called clipping, and the pieces that lie outside the frustum are discarded as there is no place to draw them.

Backface culling

Since meshes are hollow shells, not solid objects, the back side of some faces, or polygons, in the mesh will never face the camera. Typically, there is no reason to draw such faces. This is responsible for the effect often seen in computer and video games in which, if the camera happens to be inside a mesh, rather than seeing the "inside" surfaces of the mesh, it mostly disappears. (Some game engines continue to render any forward-facing or double-sided polygons, resulting in stray shapes appearing without the rest of the penetrated mesh.)

Contribution culling

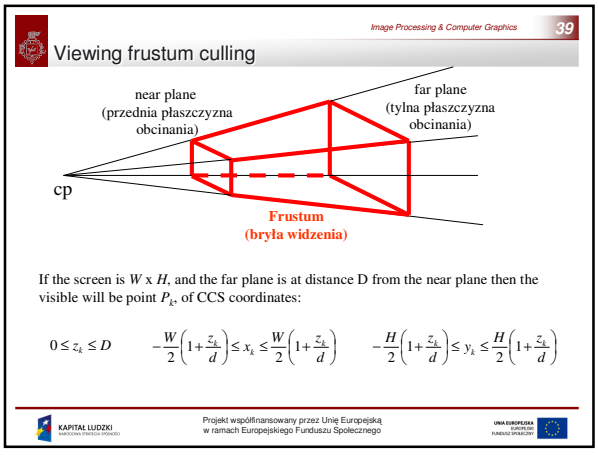
Often, objects are so far away that they do not contribute significantly to the final image. These objects are thrown away if their screen projection is too small. See Clipping plane

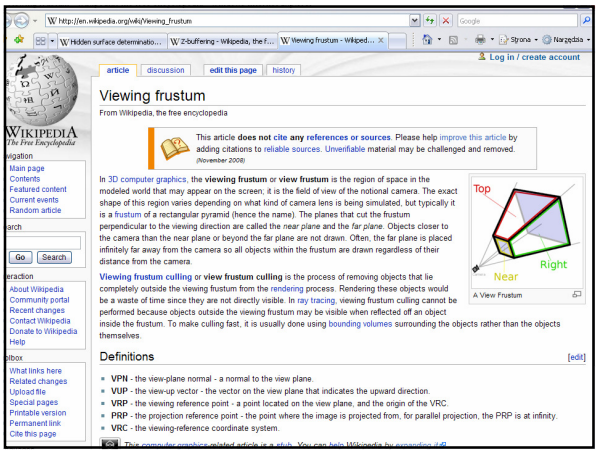
Occlusion culling

Objects that are entirely behind other opaque objects may be culled. This is a very popular mechanism to speed up the rendering of large scenes that have a moderate to high depth complexity. There are several types of occlusion culling approaches:

- Potentially visible set or *PVS* rendering, divides a scene into regions and pre-computes visibility for them. These visibility sets are then indexed at run-time to obtain high quality visibility sets (accounting for complex occluder interactions) quickly.
- Portal rendering divides a scene into cells/sectors (rooms) and portals (doors), and computes which sectors are visible by clipping them against portals.

Hansong Zhang's dissertation "Effective Occlusion Culling for the Interactive Display of Arbitrary Models" describes an occlusion culling





Viewing frustum

From Wikipedia, the free encyclopedia

This article **does not cite any references or sources**. Please help improve this article by adding citations to reliable sources. Unverifiable material may be challenged and removed. (November 2008)

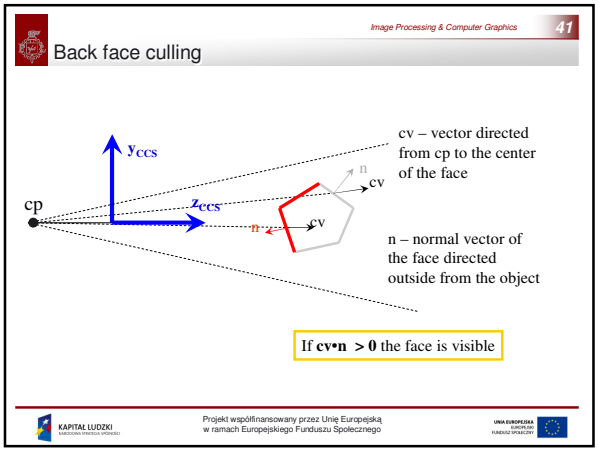
In 3D computer graphics, the **viewing frustum** or **view frustum** is the region of space in the modeled world that may appear on the screen; it is the field of view of the notional camera. The exact shape of this region varies depending on what kind of camera lens is being simulated, but typically it is a frustum of a rectangular pyramid (hence the name). The planes that cut the frustum perpendicular to the viewing direction are called the *near plane* and the *far plane*. Objects closer to the camera than the near plane or beyond the far plane are not drawn. Often, the far plane is placed infinitely far away from the camera so all objects within the frustum are drawn regardless of their distance from the camera.

Top
Right
Near
A View Frustum

Viewing frustum culling or **view frustum culling** is the process of removing objects that lie completely outside the viewing frustum from the rendering process. Rendering these objects would be a waste of time since they are not directly visible. In ray tracing, viewing frustum culling cannot be performed because objects outside the viewing frustum may be visible when reflected off an object inside the frustum. To make culling fast, it is usually done using bounding volumes surrounding the objects rather than the objects themselves.

Definitions

- **VPN** - the view-plane normal - a normal to the view plane.
- **VUP** - the view-up vector - the vector on the view plane that indicates the upward direction.
- **VRP** - the viewing reference point - a point located on the view plane, and the origin of the VRC.
- **PRP** - the projection reference point - the point where the image is projected from, for parallel projection, the PRP is at infinity.
- **VRC** - the viewing-reference coordinate system.



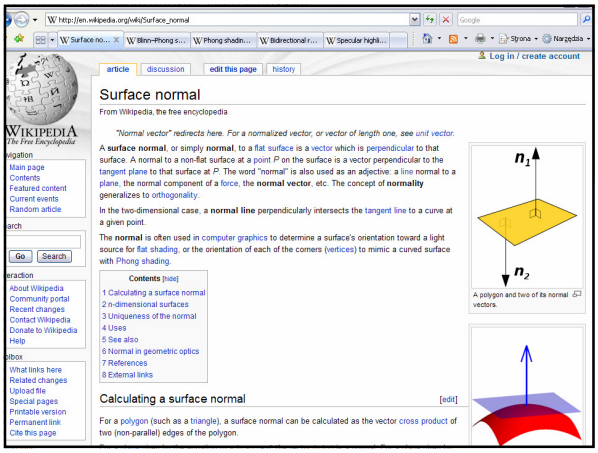
Back face culling

cv – vector directed from cp to the center of the face

n – normal vector of the face directed outside from the object

If **cv•n > 0** the face is visible

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego



Surface normal

From Wikipedia, the free encyclopedia

"Normal vector" redirects here. For a normalized vector, or vector of length one, see unit vector.

A **surface normal**, or simply **normal**, to a flat surface is a vector which is perpendicular to that surface. A normal to a non-flat surface at a point *P* on the surface is a vector perpendicular to the tangent plane to that surface at *P*. The word "normal" is also used as an adjective: a line normal to a plane, the normal component of a force, the **normal vector**, etc. The concept of **normality** generalizes to orthogonality.

In the two-dimensional case, a **normal line** perpendicularly intersects the tangent line to a curve at a given point.

The **normal** is often used in computer graphics to determine a surface's orientation toward a light source for flat shading, or the orientation of each of the corners (vertices) to mimic a curved surface with Phong shading.

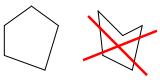
n_1
n_2
A polygon and two of its normal vectors.

Contents [hide]
1 Calculating a surface normal
2 n-dimensional surfaces
3 Uniqueness of the normal
4 Uses
5 See also
6 Normal in geometric optics
7 References
8 External links

Calculating a surface normal

For a polygon (such as a triangle), a surface normal can be calculated as the vector cross product of two (non-parallel) edges of the polygon.

Normal vectors

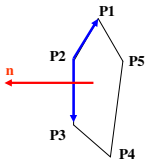


If the face is

1. coplanar and convex
2. vertices are listed in clockwise order, when looking from the inside of the object

then:

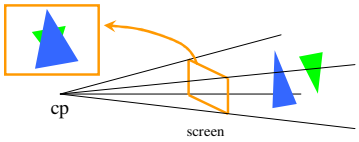
$n = (p_2p_3 \times p_2p_1) / || p_2p_3 \times p_2p_1 ||$



KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Occlusions

The closer object occludes the farther one.



- **Z-buffer (or depth buffer)**– a image size matrix for storage distance information for projected points (pixels). When image is rendered the Z-buffer is filled with distances for pixels being draw. Pixel is redraw only when distance to projected point is closer then a distance stored in Z-buffer.

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Z-buffering

From Wikipedia, the free encyclopedia

In computer graphics, **z-buffering** is the management of image depth coordinates in three-dimensional (3-D) graphics, usually done in hardware, sometimes in software. It is one solution to the visibility problem, which is the problem of deciding which elements of a rendered scene are visible, and which are hidden. The painter's algorithm is another common solution which, though less efficient, can also handle non-opaque scene elements. Z-buffering is also known as **depth buffering**.

When an object is rendered by a 3D graphics card, the depth of a generated pixel (z coordinate) is stored in a buffer (the **z-buffer** or **depth buffer**). This buffer is usually arranged as a two-dimensional array (x-y) with one element for each screen pixel. If another object of the scene must be rendered in the same pixel, the graphics card compares the two depths and chooses the one closer to the observer. The chosen depth is then saved to the z-buffer, replacing the old one. In the end, the z-buffer will allow the graphics card to correctly reproduce the usual depth perception: a close object hides a farther one. This is called **z-culling**.

The granularity of a z-buffer has a great influence on the scene quality: a 16-bit z-buffer can result in artifacts (called "z-fighting") when two objects are very close to each other. A 24-bit or 32-bit z-buffer behaves much better, although the problem cannot be entirely eliminated without additional algorithms. An 8-bit z-buffer is almost never used since it has too little precision.

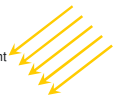
Light and shading

- Light sources
- Shading models
- Object interactions

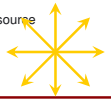
KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Light source

- Ambient light

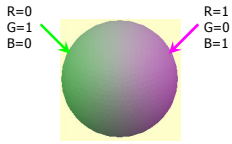


- Directed light



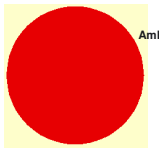
- Point light source

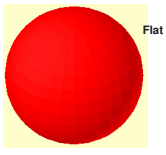
- Light color (RGB model)

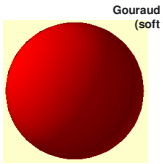


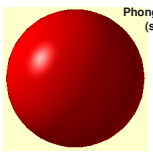
KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Modele odbić





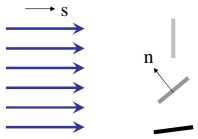




KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Shading models (flat shading)

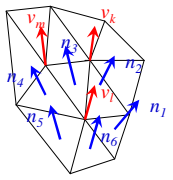
- Directed light
- Face is uniform



- **s** is a normalized light vector
- **n** is a face normal vector
- $L_R L_G L_B$ are color light components
- $J_R J_G J_B$ are color components of fase surface
- Face color is given the folowing equation:

$$- \begin{bmatrix} L_R J_R \\ L_G J_G \\ L_B J_B \end{bmatrix} n \bullet s$$

Gouraud shading model

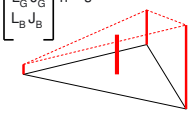


- Normal vectors at vertices are estimated as average of normal vectors of connected faces

$$v_i = (n_1 + n_2 + \ldots + n_i)/i$$

- Vector **n'** is interpolated over the face area by means of bilinear interpolation
- Face color may vary over the face area

$$- \begin{bmatrix} L_R J_R \\ L_G J_G \\ L_B J_B \end{bmatrix} n' \bullet s$$



- Triangular mesh
- Directed light or point source light

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

article | discussion | edit this page | history

Log in / create account

Gouraud shading

From Wikipedia, the free encyclopedia

This article includes a list of references or external links, but its sources remain unclear because it lacks inline citations. Please improve this article by introducing more precise citations where appropriate. (October 2008)

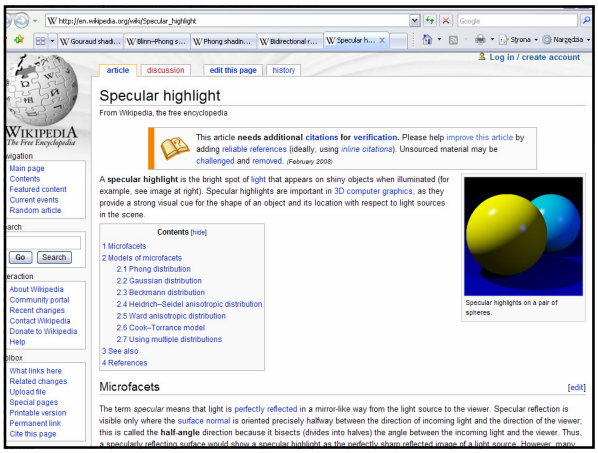
Gouraud shading, named after Henri Gouraud, is a method used in computer graphics to simulate the differing effects of light and colour across the surface of an object. In practice, Gouraud shading is used to achieve smooth lighting on low-polygon surfaces without the heavy computational requirements of calculating lighting for each pixel. Gouraud first published the technique in 1971.

Gouraud-shaded polygon.

Principles behind the method [edit]

The basic principle behind the method is as follows: An estimate to the surface normal of each vertex in a 3D model is found by averaging the surface normals of polygons which meet at each vertex. Using these estimates, lighting computations based on the Phong reflection model are then performed to produce colour intensities at the vertices. Screen pixel intensities can then be bilinearly interpolated from the colour values calculated at the vertices. In a simpler sense, a gradient is formed in the area where the vertices meet.

Gouraud shading's strength and weakness lies in its interpolation. Interpolating colour values from three samples of expensive lighting calculations is much less processor intensive than performing that lighting calculation for each pixel, as is done in Phong shading. However, highly localized lighting effects (such as specular highlights, e.g. the glint of reflected light on the surface of an apple) will not be rendered correctly, and if a highlight lies in the middle of a polygon, but does not spread to the polygon's vertex, it will not be apparent in a Gouraud rendering; conversely, if a highlight occurs at the vertex of a polygon, it will be rendered correctly at this vertex (as this is where the lighting model is applied), but will be spread unnaturally across all neighboring polygons via the interpolation method. The problem is easily spotted in a rendering which ought to have a specular highlight moving smoothly across the surface of a model as it rotates. Gouraud shading will instead produce a highlight continuously fading in and out across neighboring portions of the model, peaking in intensity when the intended specular highlight passes over a vertex of the model. (This can be improved by increasing the



Specular highlight

From Wikipedia, the free encyclopedia

This article needs additional citations for verification. Please help improve this article by adding reliable references (ideally, using inline citations). Unsourced material may be challenged and removed. (February 2008)

A **specular highlight** is the bright spot of light that appears on shiny objects when illuminated (for example, see image at right). Specular highlights are important in 3D computer graphics, as they provide a strong visual cue for the shape of an object and its location with respect to light sources in the scene.

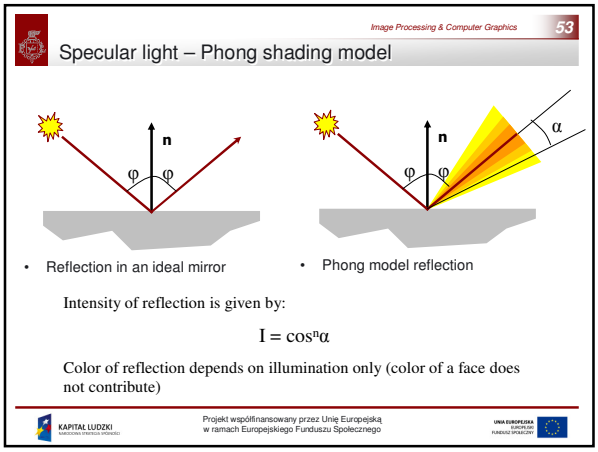
Specular highlights on a pair of spheres.

Contents [hide]

1 Microfacets
2 Models of microfacets
2.1 Phong distribution
2.2 Gaussian distribution
2.3 Beckmann distribution
2.4 Heidrich–Seidel anisotropic distribution
2.5 Ward anisotropic distribution
2.6 Cook–Torrance model
2.7 Using multiple distributions
3 See also
4 References

Microfacets [edit]

The term *specular* means that light is perfectly reflected in a mirror-like way from the light source to the viewer. Specular reflection is visible only where the surface normal is oriented precisely halfway between the direction of incoming light and the direction of the viewer; this is called the **half-angle** direction because it bisects (divides into halves) the angle between the incoming light and the viewer. Thus, a specularly reflecting surface would show a specular highlight as the perfectly sharp reflected image of a light source. However, many



Specular light – Phong shading model

- Reflection in an ideal mirror

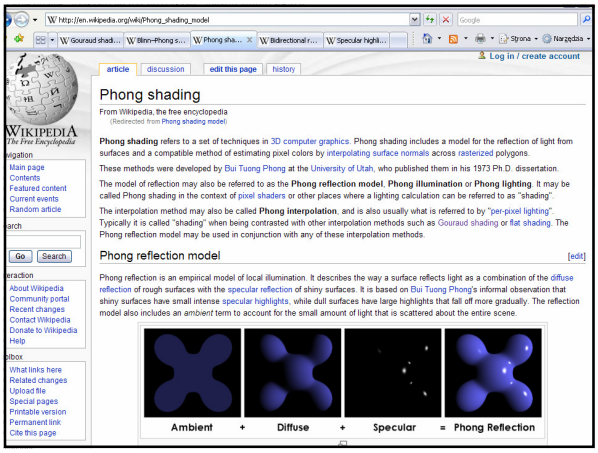
- Phong model reflection

Intensity of reflection is given by:

$$I = \cos^n\alpha$$

Color of reflection depends on illumination only (color of a face does not contribute)

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY



Phong shading

From Wikipedia, the free encyclopedia
(Redirected from Phong shading model)

Phong shading refers to a set of techniques in 3D computer graphics. Phong shading includes a model for the reflection of light from surfaces and a compatible method of estimating pixel colors by interpolating surface normals across rasterized polygons.

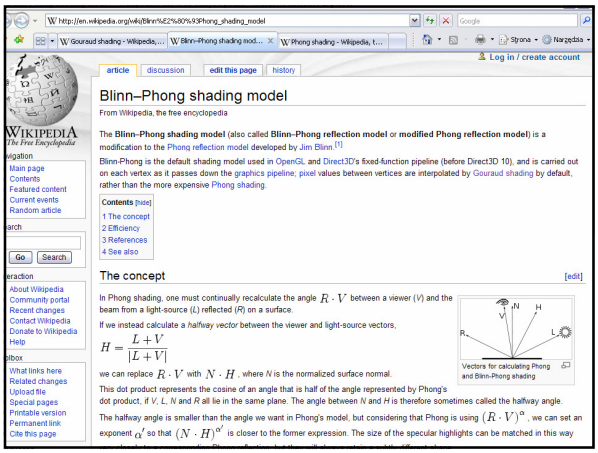
These methods were developed by Bui Tuong Phong at the University of Utah, who published them in his 1973 Ph.D. dissertation.

The model of reflection may also be referred to as the **Phong reflection model**, **Phong illumination** or **Phong lighting**. It may be called Phong shading in the context of pixel shaders or other places where a lighting calculation can be referred to as "shading".

The interpolation method may also be called **Phong interpolation**, and is also usually what is referred to by "per-pixel lighting". Typically it is called "shading" when being contrasted with other interpolation methods such as Gouraud shading or flat shading. The Phong reflection model may be used in conjunction with any of these interpolation methods.

Phong reflection model [edit]

Phong reflection is an empirical model of local illumination. It describes the way a surface reflects light as a combination of the diffuse reflection of rough surfaces with the specular reflection of shiny surfaces. It is based on Bui Tuong Phong's informal observation that shiny surfaces have small intense specular highlights, while dull surfaces have large highlights that fall off more gradually. The reflection model also includes an *ambient* term to account for the small amount of light that is scattered about the entire scene.



Blinn–Phong shading model

From Wikipedia, the free encyclopedia

The **Blinn-Phong shading model** (also called **Blinn-Phong reflection model** or **modified Phong reflection model**) is a modification to the Phong reflection model developed by Jim Blinn.[1]

Blinn-Phong is the default shading model used in OpenGL and Direct3D's fixed-function pipeline (before Direct3D 10), and is carried out on each vertex as it passes down the graphics pipeline; pixel values between vertices are interpolated by Gouraud shading by default, rather than the more expensive Phong shading.

Contents [hide]

1 The concept
2 Efficiency
3 References
4 See also

The concept

In Phong shading, one must continually recalculate the angle $R \cdot V$ between a viewer (V) and the beam from a light-source (L) reflected (R) on a surface.

If we instead calculate a *halfway vector* between the viewer and light-source vectors,

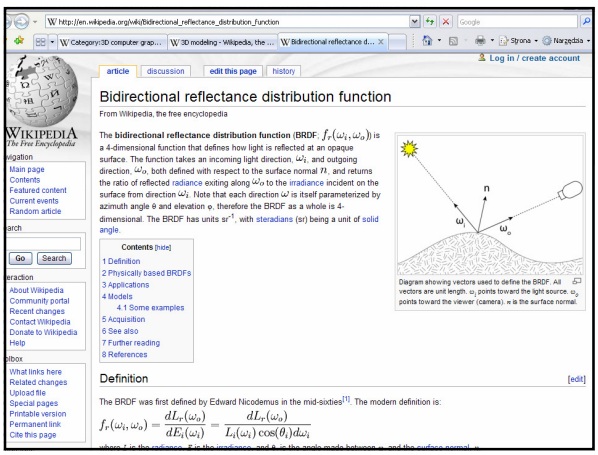
$$H = \frac{L+V}{|L+V|}$$

we can replace $R \cdot V$ with $N \cdot H$, where N is the normalized surface normal.

This dot product represents the cosine of an angle that is half of the angle represented by Phong's dot product, if V, L, N and R all lie in the same plane. The angle between N and H is therefore sometimes called the halfway angle.

The halfway angle is smaller than the angle we want in Phong's model, but considering that Phong is using $(R \cdot V)^{\alpha}$, we can set an exponent α' so that $(N \cdot H)^{\alpha'}$ is closer to the former expression. The size of the specular highlights can be matched in this way

Vectors for calculating Phong and Blinn-Phong shading



Bidirectional reflectance distribution function

From Wikipedia, the free encyclopedia

The **bidirectional reflectance distribution function** (**BRDF**), $f_r(\omega_i, \omega_o)$ is a 4-dimensional function that defines how light is reflected at an opaque surface. The function takes an incoming light direction, ω_i, and outgoing direction, ω_o, both defined with respect to the surface normal n, and returns the ratio of reflected radiance exiting along ω_o to the irradiance incident on the surface from direction ω_i. Note that each direction ω is itself parameterized by azimuth angle ϕ and elevation φ, therefore the BRDF as a whole is 4-dimensional. The BRDF has units sr^{-1}, with steradians (sr) being a unit of solid angle.

Contents [hide]

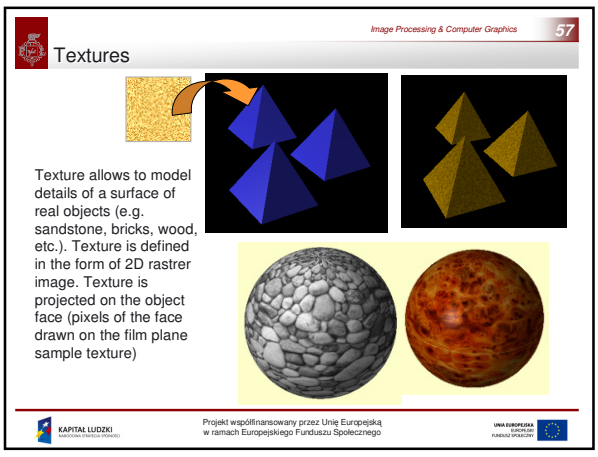
1 Definition
2 Physically based BRDFs
3 Applications
4 Models
4.1 Some examples
5 Acquisition
6 See also
7 Further reading
8 References

Diagram showing vectors used to define the BRDF. All vectors are unit length. ω_i points toward the light source, ω_o points toward the viewer (camera), n is the surface normal.

Definition

The BRDF was first defined by Edward Nicodemus in the mid-sixties[1]. The modern definition is:

$$f_r(\omega_i, \omega_o) = \frac{dL_r(\omega_o)}{dE_i(\omega_i)} = \frac{dL_r(\omega_o)}{L_i(\omega_i)\cos(\theta_i)d\omega_i}$$



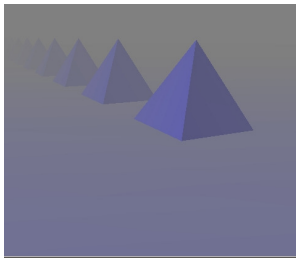
Textures

Texture allows to model details of a surface of real objects (e.g. sandstone, bricks, wood, etc.). Texture is defined in the form of 2D rastrer image. Texture is projected on the object face (pixels of the face drawn on the film plane sample texture)

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Fog

The further object the less visible in the fog. The fog model blends the fog color and shaded scene.



$k = \alpha_m k_{scene} + (1-\alpha_m) k_{fog}$

where $\alpha_m = f(z)$

$f(z)$ – fog modeling function

Linear function

$f(\mathbf{z}) = 1$, for $z < z_p$

$= 0$, for $z > z_k$

$= (z_k - \mathbf{z})/(z_k - z_p)$, for others z

Exponent function

$f(\mathbf{z}) = \exp(-g\mathbf{z})$

Square exponent function

$f(\mathbf{z}) = \exp(-g\mathbf{z}^2)$

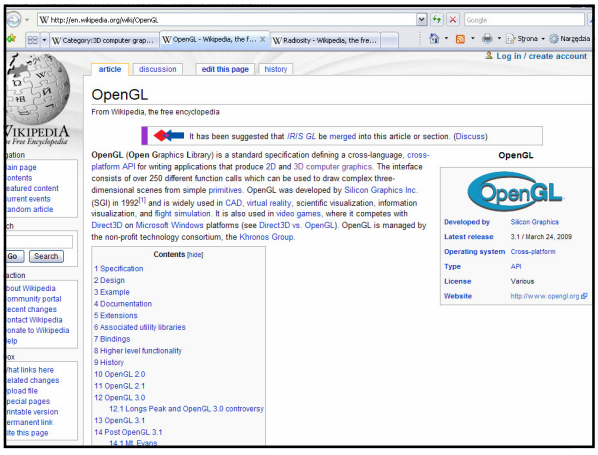
z – object distance from camera

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

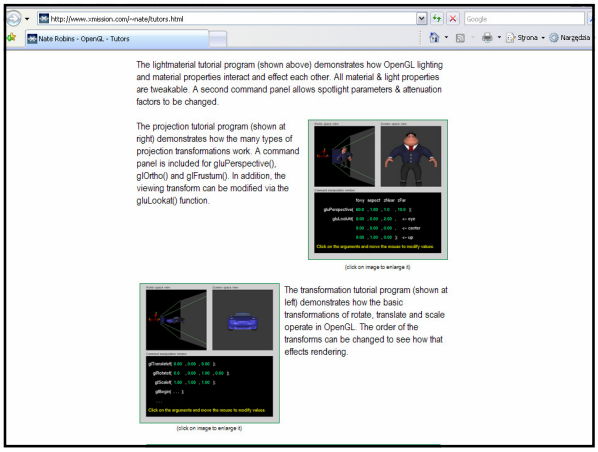
APIs for 3D graphics

- OpenGL (*GL - Graphics Language*)
  - standard and algorithms developed by Silicon Graphics for IRIS workstation,
  - specification 1.0 in 1992 (multiplatform standard),
  - Microsoft included OpenGL in Windows NT 4.0 and 95,
- Direct 3D (Direct X compont)
  - Reality Lab from RenderMorphing (in 1994),
  - Microsoft buys RenderMorphing (1995) and creates Direct3D based on Reality Lab,
  - Direct X works on Windows only.

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY




http://www.xmission.com/~nate/tutors.html
Nate Robins - OpenGL - Tutors
nate robins
main OPENGL CHRONICLES ALLIES
Tutors
Tutors source code package (tutors-src.zip 920KB)
Source package + Windows executables (tutors-win32.zip 1.24MB)
Source package + Mac OS X universal binaries (tutors-macosx.zip 1.2MB)
Contributions from various people (Unix makefiles, etc).
The fog tutorial program (shown at right) demonstrates how to use depth cueing (or fog) in OpenGL. It includes command panels that allow dynamic update of the parameters for linear, exponential and exponential2 fog functions. The fog color can also be dynamically changed.


http://www.xmission.com/~nate/tutors.html
Nate Robins - OpenGL - Tutors
The lightmaterial tutorial program (shown above) demonstrates how OpenGL lighting and material properties interact and effect each other. All material & light properties are tweakable. A second command panel allows spotlight parameters & attenuation factors to be changed.
The projection tutorial program (shown at right) demonstrates how the many types of projection transformations work. A command panel is included for gluPerspective(), glOrtho() and glFrustum(). In addition, the viewing transform can be modified via the gluLookat() function.
(click on image to enlarge it)
The transformation tutorial program (shown at left) demonstrates how the basic transformations of rotate, translate and scale operate in OpenGL. The order of the transforms can be changed to see how that effects rendering.
(click on image to enlarge it)

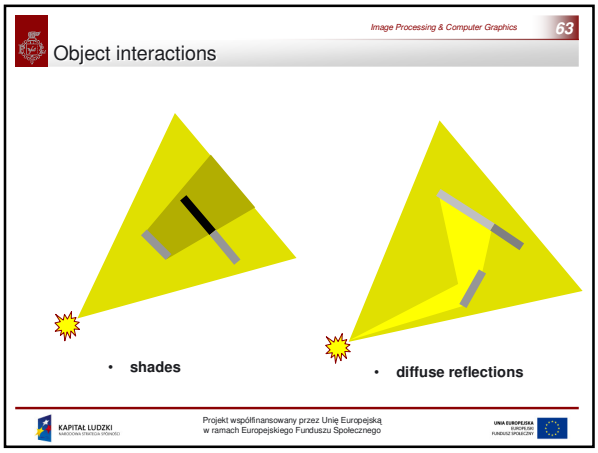
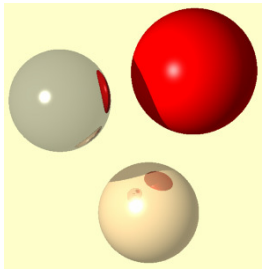
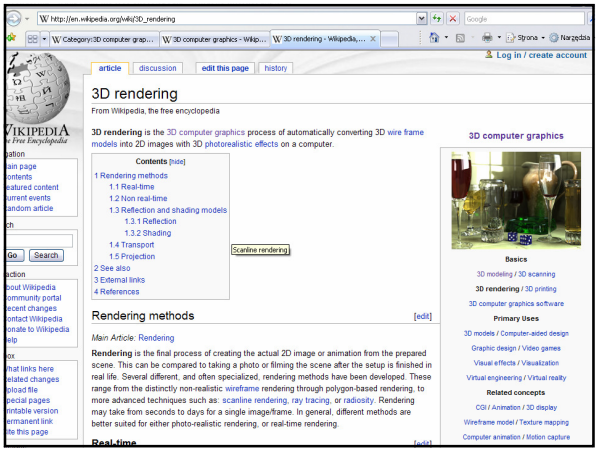


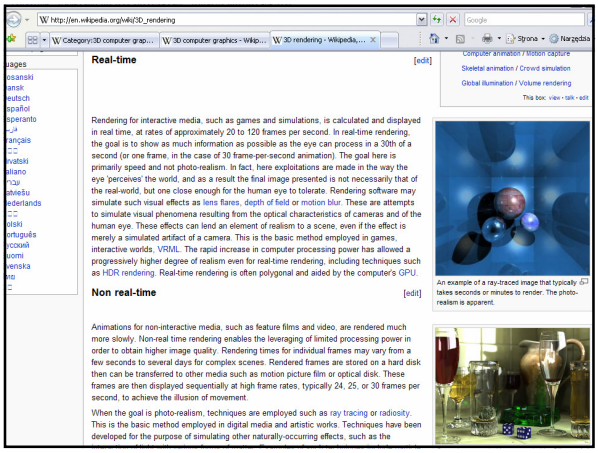
Image Processing & Computer Graphics
63
Object interactions
• shades
• diffuse reflections
KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

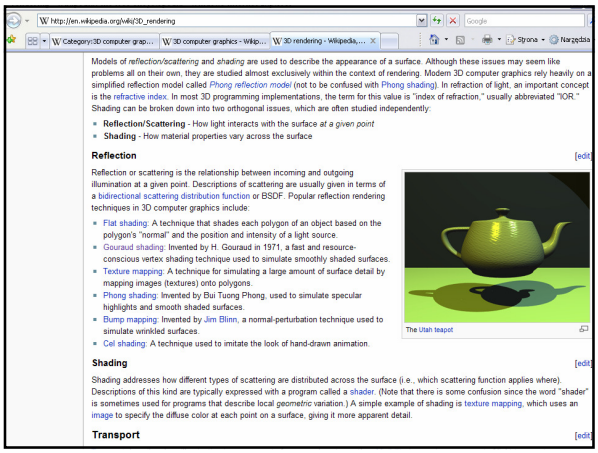
Ray tracing techniques



KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY







Models of *reflection/scattering* and *shading* are used to describe the appearance of a surface. Although these issues may seem like problems all on their own, they are studied almost exclusively within the context of rendering. Modern 3D computer graphics rely heavily on a simplified reflection model called *Phong reflection model* (not to be confused with Phong shading). In refraction of light, an important concept is the refractive index. In most 3D programming implementations, the term for this value is "index of refraction," usually abbreviated "IOR." Shading can be broken down into two orthogonal issues, which are often studied independently:

- **Reflection/Scattering** - How light interacts with the surface *at a given point*
- **Shading** - How material properties vary across the surface

Reflection

Reflection or scattering is the relationship between incoming and outgoing illumination at a given point. Descriptions of scattering are usually given in terms of a bidirectional scattering distribution function or BSDF. Popular reflection rendering techniques in 3D computer graphics include:

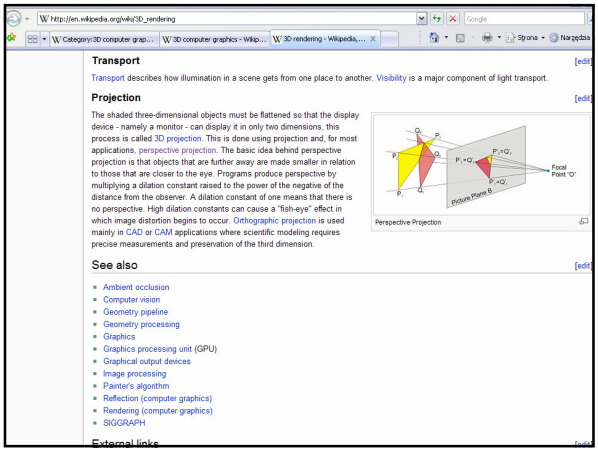
- Flat shading: A technique that shades each polygon of an object based on the polygon's "normal" and the position and intensity of a light source.
- Gouraud shading: Invented by H. Gouraud in 1971, a fast and resource-conscious vertex shading technique used to simulate smoothly shaded surfaces.
- Texture mapping: A technique for simulating a large amount of surface detail by mapping images (textures) onto polygons.
- Phong shading: Invented by Bui Tuong Phong, used to simulate specular highlights and smooth shaded surfaces.
- Bump mapping: Invented by Jim Blinn, a normal-perturbation technique used to simulate wrinkled surfaces.
- Cel shading: A technique used to imitate the look of hand-drawn animation.

The Utah teapot

Shading

Shading addresses how different types of scattering are distributed across the surface (i.e., which scattering function applies where). Descriptions of this kind are typically expressed with a program called a shader. (Note that there is some confusion since the word "shader" is sometimes used for programs that describe local *geometric* variation.) A simple example of shading is texture mapping, which uses an image to specify the diffuse color at each point on a surface, giving it more apparent detail.

Transport



Transport

Transport describes how illumination in a scene gets from one place to another. Visibility is a major component of light transport.

Projection

The shaded three-dimensional objects must be flattened so that the display device - namely a monitor - can display it in only two dimensions, this process is called 3D projection. This is done using projection and, for most applications, perspective projection. The basic idea behind perspective projection is that objects that are further away are made smaller in relation to those that are closer to the eye. Programs produce perspective by multiplying a dilation constant raised to the power of the negative of the distance from the observer. A dilation constant of one means that there is no perspective. High dilation constants can cause a "fish-eye" effect in which image distortion begins to occur. Orthographic projection is used mainly in CAD or CAM applications where scientific modeling requires precise measurements and preservation of the third dimension.

Perspective Projection

See also

- Ambient occlusion
- Computer vision
- Geometry pipeline
- Geometry processing
- Graphics
- Graphics processing unit (GPU)
- Graphical output devices
- Image processing
- Painter's algorithm
- Reflection (computer graphics)
- Rendering (computer graphics)
- SIGGRAPH

External links

article | discussion | edit this page | history

Log in / create account

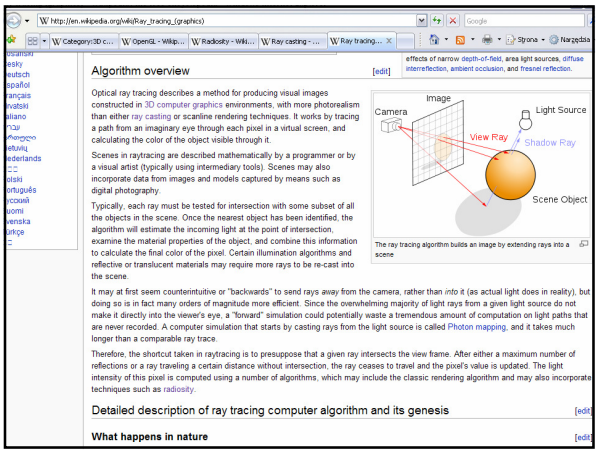
Ray tracing (graphics)

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification.** Please help improve this article by adding reliable references (ideally, using *inline citations*). Unsourced material may be challenged and removed. (March 2008)

In computer graphics, **ray tracing** is a technique for generating an image by tracing the path of light through pixels in an image plane. The technique is capable of producing a very high degree of photorealism; usually higher than that of typical scanline rendering methods, but at a greater computational cost. This makes ray tracing best suited for applications where the image can be rendered slowly ahead of time, such as in still images and film and television special effects, and more poorly suited for real-time applications like computer games where speed is critical. Ray tracing is capable of simulating a wide variety of optical effects, such as reflection and refraction, scattering, and chromatic aberration.

Contents [hide]



Algorithm overview

Optical ray tracing describes a method for producing visual images constructed in 3D computer graphics environments, with more photorealism than either ray casting or scanline rendering techniques. It works by tracing a path from an imaginary eye through each pixel in a virtual screen, and calculating the color of the object visible through it.

Scenes in raytracing are described mathematically by a programmer or by a visual artist (typically using intermediary tools). Scenes may also incorporate data from images and models captured by means such as digital photography.

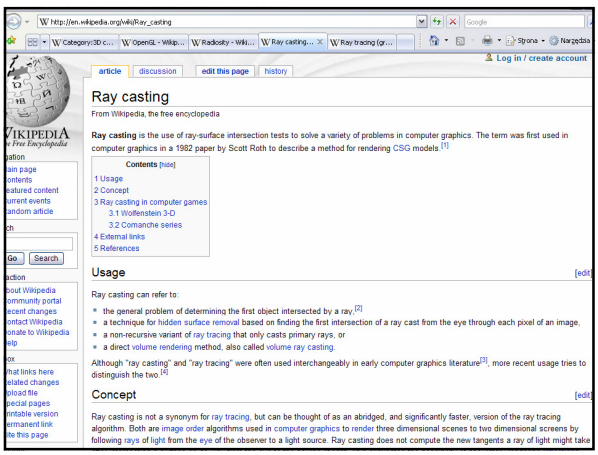
Typically, each ray must be tested for intersection with some subset of all the objects in the scene. Once the nearest object has been identified, the algorithm will estimate the incoming light at the point of intersection, examine the material properties of the object, and combine this information to calculate the final color of the pixel. Certain illumination algorithms and reflective or translucent materials may require more rays to be re-cast into the scene.

It may at first seem counterintuitive or "backwards" to send rays away from the camera, rather than *into* it (as actual light does in reality), but doing so is in fact many orders of magnitude more efficient. Since the overwhelming majority of light rays from a given light source do not make it directly into the viewer's eye, a "forward" simulation could potentially waste a tremendous amount of computation on light paths that are never recorded. A computer simulation that starts by casting rays from the light source is called Photon mapping, and it takes much longer than a comparable ray trace.

Therefore, the shortcut taken in raytracing is to presuppose that a given ray intersects the view frame. After either a maximum number of reflections or a ray traveling a certain distance without intersection, the ray ceases to travel and the pixel's value is updated. The light intensity of this pixel is computed using a number of algorithms, which may include the classic rendering algorithm and may also incorporate techniques such as radiosity.

Detailed description of ray tracing computer algorithm and its genesis

What happens in nature



Ray casting

From Wikipedia, the free encyclopedia

Ray casting is the use of ray-surface intersection tests to solve a variety of problems in computer graphics. The term was first used in computer graphics in a 1982 paper by Scott Roth to describe a method for rendering CSG models.[1]

Contents [hide]

1 Usage
2 Concept
3 Ray casting in computer games
3.1 Wolfenstein 3-D
3.2 Comanche series
4 External links
5 References

Usage

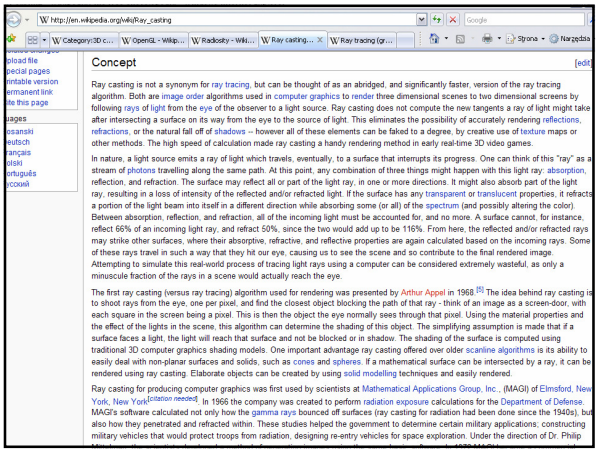
Ray casting can refer to:

- the general problem of determining the first object intersected by a ray,[2]
- a technique for hidden surface removal based on finding the first intersection of a ray cast from the eye through each pixel of an image,
- a non-recursive variant of ray tracing that only casts primary rays, or
- a direct volume rendering method, also called volume ray casting.

Although "ray casting" and "ray tracing" were often used interchangeably in early computer graphics literature[3], more recent usage tries to distinguish the two.[4]

Concept

Ray casting is not a synonym for ray tracing, but can be thought of as an abridged, and significantly faster, version of the ray tracing algorithm. Both are image order algorithms used in computer graphics to render three dimensional scenes to two dimensional screens by following rays of light from the eye of the observer to a light source. Ray casting does not compute the new tangents a ray of light might take



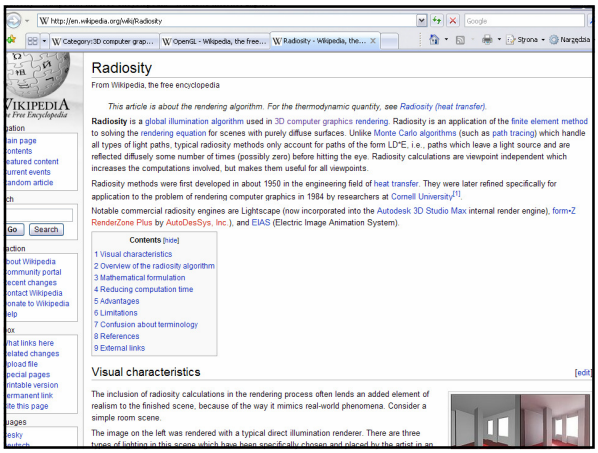
Concept

Ray casting is not a synonym for ray tracing, but can be thought of as an abridged, and significantly faster, version of the ray tracing algorithm. Both are image order algorithms used in computer graphics to render three dimensional scenes to two dimensional screens by following rays of light from the eye of the observer to a light source. Ray casting does not compute the new tangents a ray of light might take after intersecting a surface on its way from the eye to the source of light. This eliminates the possibility of accurately rendering reflections, refractions, or the natural fall off of shadows – however all of these elements can be faked to a degree, by creative use of texture maps or other methods. The high speed of calculation made ray casting a handy rendering method in early real-time 3D video games.

In nature, a light source emits a ray of light which travels, eventually, to a surface that interrupts its progress. One can think of this "ray" as a stream of photons travelling along the same path. At this point, any combination of three things might happen with this light ray: absorption, reflection, and refraction. The surface may reflect all or part of the light ray, in one or more directions. It might also absorb part of the light ray, resulting in a loss of intensity of the reflected and/or refracted light. If the surface has any transparent or translucent properties, it refracts a portion of the light beam into itself in a different direction while absorbing some (or all) of the spectrum (and possibly altering the color). Between absorption, reflection, and refraction, all of the incoming light must be accounted for, and no more. A surface cannot, for instance, reflect 66% of an incoming light ray, and refract 50%, since the two would add up to be 116%. From here, the reflected and/or refracted rays may strike other surfaces, where their absorptive, refractive, and reflective properties are again calculated based on the incoming rays. Some of these rays travel in such a way that they hit our eye, causing us to see the scene and so contribute to the final rendered image. Attempting to simulate this real-world process of tracing light rays using a computer can be considered extremely wasteful, as only a minuscule fraction of the rays in a scene would actually reach the eye.

The first ray casting (versus ray tracing) algorithm used for rendering was presented by Arthur Appel in 1968.[5] The idea behind ray casting is to shoot rays from the eye, one per pixel, and find the closest object blocking the path of that ray - think of an image as a screen-door, with each square in the screen being a pixel. This is then the object the eye normally sees through that pixel. Using the material properties and the effect of the lights in the scene, this algorithm can determine the shading of this object. The simplifying assumption is made that if a surface faces a light, the light will reach that surface and not be blocked or in shadow. The shading of the surface is computed using traditional 3D computer graphics shading models. One important advantage ray casting offered over older scanline algorithms is its ability to easily deal with non-planar surfaces and solids, such as cones and spheres. If a mathematical surface can be intersected by a ray, it can be rendered using ray casting. Elaborate objects can be created by using solid modelling techniques and easily rendered.

Ray casting for producing computer graphics was first used by scientists at Mathematical Applications Group, Inc., (MAGI) of Elmsford, New York, New York[citation needed]. In 1966 the company was created to perform radiation exposure calculations for the Department of Defense. MAGI's software calculated not only how the gamma rays bounced off surfaces (ray casting for radiation had been done since the 1940s), but also how they penetrated and refracted within. These studies helped the government to determine certain military applications; constructing military vehicles that would protect troops from radiation, designing re-entry vehicles for space exploration. Under the direction of Dr. Philip



Radiosity

From Wikipedia, the free encyclopedia

This article is about the rendering algorithm. For the thermodynamic quantity, see Radiosity (heat transfer).

Radiosity is a global illumination algorithm used in 3D computer graphics rendering. Radiosity is an application of the finite element method to solving the rendering equation for scenes with purely diffuse surfaces. Unlike Monte Carlo algorithms (such as path tracing) which handle all types of light paths, typical radiosity methods only account for paths of the form LD*E, i.e., paths which leave a light source and are reflected diffusely some number of times (possibly zero) before hitting the eye. Radiosity calculations are viewpoint independent which increases the computations involved, but makes them useful for all viewpoints.

Radiosity methods were first developed in about 1950 in the engineering field of heat transfer. They were later refined specifically for application to the problem of rendering computer graphics in 1984 by researchers at Cornell University.[1]

Notable commercial radiosity engines are Lightscape (now incorporated into the Autodesk 3D Studio Max internal render engine), form-Z RenderZone Plus by AutoDesSys, Inc.), and ElAS (Electric Image Animation System).

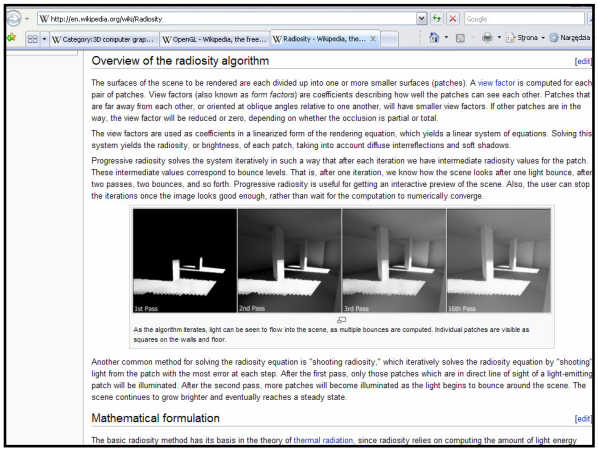
Contents [hide]

1 Visual characteristics
2 Overview of the radiosity algorithm
3 Mathematical formulation
4 Reducing computation time
5 Advantages
6 Limitations
7 Confusion about terminology
8 References
9 External links

Visual characteristics [edit]

The inclusion of radiosity calculations in the rendering process often lends an added element of realism to the finished scene, because of the way it mimics real-world phenomena. Consider a simple room scene.

The image on the left was rendered with a typical direct illumination renderer. There are three



Overview of the radiosity algorithm [edit]

The surfaces of the scene to be rendered are each divided up into one or more smaller surfaces (patches). A view factor is computed for each pair of patches. View factors (also known as *form factors*) are coefficients describing how well the patches can see each other. Patches that are far away from each other, or oriented at oblique angles relative to one another, will have smaller view factors. If other patches are in the way, the view factor will be reduced or zero, depending on whether the occlusion is partial or total.

The view factors are used as coefficients in a linearized form of the rendering equation, which yields a linear system of equations. Solving this system yields the radiosity, or brightness, of each patch, taking into account diffuse interreflections and soft shadows.

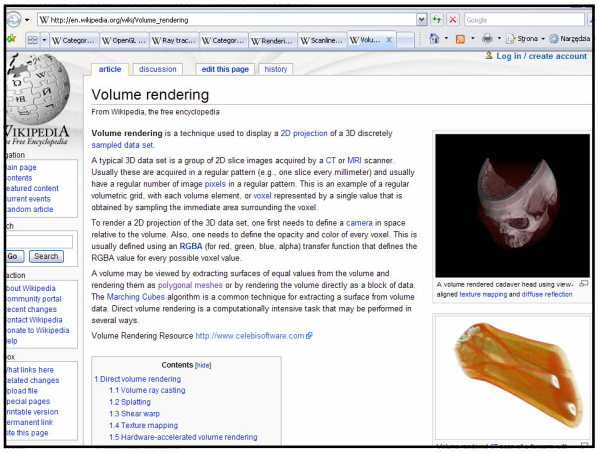
Progressive radiosity solves the system iteratively in such a way that after each iteration we have intermediate radiosity values for the patch. These intermediate values correspond to bounce levels. That is, after one iteration, we know how the scene looks after one light bounce, after two passes, two bounces, and so forth. Progressive radiosity is useful for getting an interactive preview of the scene. Also, the user can stop the iterations once the image looks good enough, rather than wait for the computation to numerically converge.

As the algorithm iterates, light can be seen to flow into the scene, as multiple bounces are computed. Individual patches are visible as squares on the walls and floor.

Another common method for solving the radiosity equation is "shooting radiosity," which iteratively solves the radiosity equation by "shooting" light from the patch with the most error at each step. After the first pass, only those patches which are in direct line of sight of a light-emitting patch will be illuminated. After the second pass, more patches will become illuminated as the light begins to bounce around the scene. The scene continues to grow brighter and eventually reaches a steady state.

Mathematical formulation [edit]

The basic radiosity method has its basis in the theory of thermal radiation, since radiosity relies on computing the amount of light energy



article discussion edit this page history

Log in / create account

Volume rendering

From Wikipedia, the free encyclopedia

Volume rendering is a technique used to display a 2D projection of a 3D discretely sampled data set.

A typical 3D data set is a group of 2D slice images acquired by a CT or MRI scanner. Usually these are acquired in a regular pattern (e.g., one slice every millimeter) and usually have a regular number of image pixels in a regular pattern. This is an example of a regular volumetric grid, with each volume element, or voxel represented by a single value that is obtained by sampling the immediate area surrounding the voxel.

To render a 2D projection of the 3D data set, one first needs to define a camera in space relative to the volume. Also, one needs to define the opacity and color of every voxel. This is usually defined using an RGBA (for red, green, blue, alpha) transfer function that defines the RGBA value for every possible voxel value.

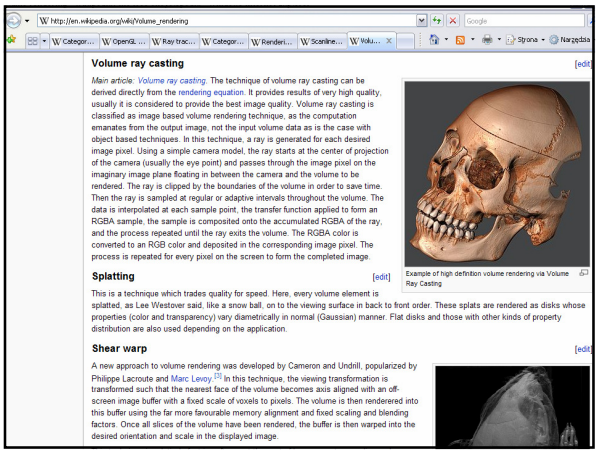
A volume may be viewed by extracting surfaces of equal values from the volume and rendering them as polygonal meshes or by rendering the volume directly as a block of data. The Marching Cubes algorithm is a common technique for extracting a surface from volume data. Direct volume rendering is a computationally intensive task that may be performed in several ways.

Volume Rendering Resource http://www.celebisoftware.com

A volume rendered cadaver head using view-aligned texture mapping and diffuse reflection

Contents [hide]

1 Direct volume rendering
1.1 Volume ray casting
1.2 Splatting
1.3 Shear warp
1.4 Texture mapping
1.5 Hardware-accelerated volume rendering



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

Volume ray casting

[edit]

Main article: Volume ray casting. The technique of volume ray casting can be derived directly from the rendering equation. It provides results of very high quality, usually it is considered to provide the best image quality. Volume ray casting is classified as image based volume rendering technique, as the computation emanates from the output image, not the input volume data as is the case with object based techniques. In this technique, a ray is generated for each desired image pixel. Using a simple camera model, the ray starts at the center of projection of the camera (usually the eye point) and passes through the image pixel on the imaginary image plane floating in between the camera and the volume to be rendered. The ray is clipped by the boundaries of the volume in order to save time. Then the ray is sampled at regular or adaptive intervals throughout the volume. The data is interpolated at each sample point, the transfer function applied to form an RGBA sample, the sample is composited onto the accumulated RGBA of the ray, and the process repeated until the ray exits the volume. The RGBA color is converted to an RGB color and deposited in the corresponding image pixel. The process is repeated for every pixel on the screen to form the completed image.

Example of high definition volume rendering via Volume Ray Casting

Splatting

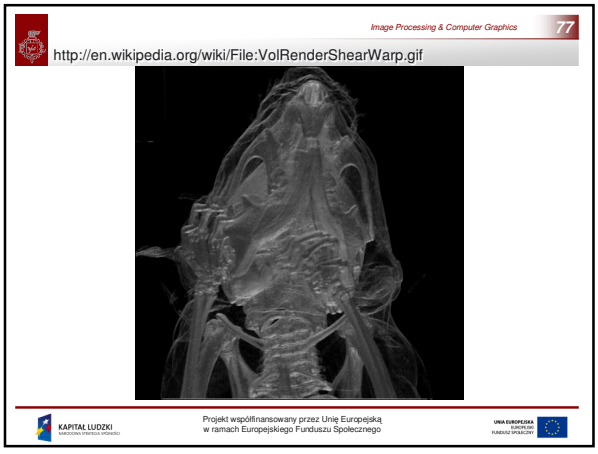
[edit]

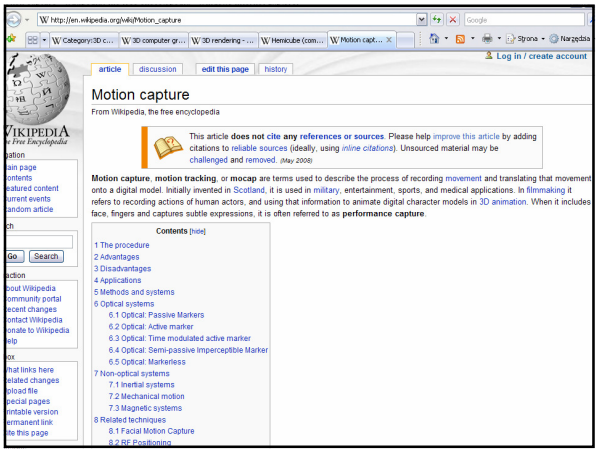
This is a technique which trades quality for speed. Here, every volume element is splatted, as Lee Westover said, like a snow ball, on to the viewing surface in back to front order. These splats are rendered as disks whose properties (color and transparency) vary diametrically in normal (Gaussian) manner. Flat disks and those with other kinds of property distribution are also used depending on the application.

Shear warp

[edit]

A new approach to volume rendering was developed by Cameron and Undrill, popularized by Philippe Lacroute and Marc Levoy.[5] In this technique, the viewing transformation is transformed such that the nearest face of the volume becomes axis aligned with an off-screen image buffer with a fixed scale of voxels to pixels. The volume is then rendered into this buffer using the far more favourable memory alignment and fixed scaling and blending factors. Once all slices of the volume have been rendered, the buffer is then warped into the desired orientation and scale in the displayed image.





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

Log in / create account

article | discussion | edit this page | history

Motion capture

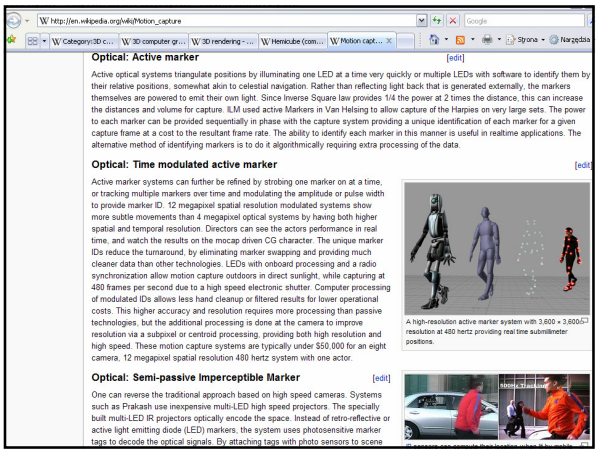
From Wikipedia, the free encyclopedia

This article **does not cite any references or sources**. Please help improve this article by adding citations to reliable sources (ideally, using *inline citations*). Unsourced material may be challenged and removed. *(May 2008)*

Motion capture, **motion tracking**, or **mocap** are terms used to describe the process of recording movement and translating that movement onto a digital model. Initially invented in Scotland, it is used in military, entertainment, sports, and medical applications. In filmmaking it refers to recording actions of human actors, and using that information to animate digital character models in 3D animation. When it includes face, fingers and captures subtle expressions, it is often referred to as **performance capture**.

Contents [hide]

1 The procedure
2 Advantages
3 Disadvantages
4 Applications
5 Methods and systems
6 Optical systems
6.1 Optical: Passive Markers
6.2 Optical: Active marker
6.3 Optical: Time modulated active marker
6.4 Optical: Semi-passive Imperceptible Marker
6.5 Optical: Markerless
7 Non-optical systems
7.1 Inertial systems
7.2 Mechanical motion
7.3 Magnetic systems
8 Related techniques
8.1 Facial Motion Capture
8.2 RF Positioning



Optical: Active marker [edit]

Active optical systems triangulate positions by illuminating one LED at a time very quickly or multiple LEDs with software to identify them by their relative positions, somewhat akin to celestial navigation. Rather than reflecting light back that is generated externally, the markers themselves are powered to emit their own light. Since Inverse Square law provides 1/4 the power at 2 times the distance, this can increase the distances and volume for capture. ILM used active Markers in Van Helsing to allow capture of the Harpies on very large sets. The power to each marker can be provided sequentially in phase with the capture system providing a unique identification of each marker for a given capture frame at a cost to the resultant frame rate. The ability to identify each marker in this manner is useful in realtime applications. The alternative method of identifying markers is to do it algorithmically requiring extra processing of the data.

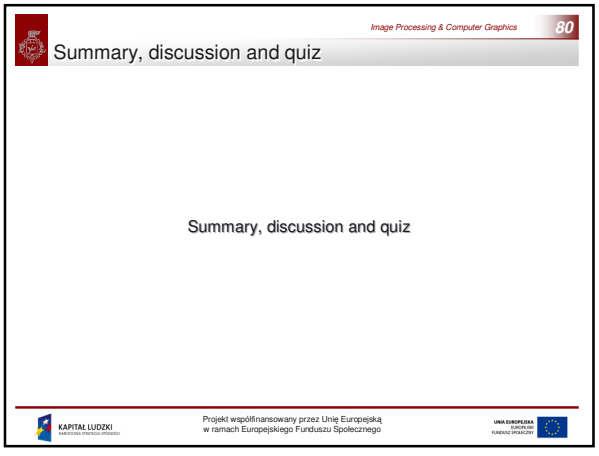
Optical: Time modulated active marker [edit]

Active marker systems can further be refined by strobing one marker on at a time, or tracking multiple markers over time and modulating the amplitude or pulse width to provide marker ID. 12 megapixel spatial resolution modulated systems show more subtle movements than 4 megapixel optical systems by having both higher spatial and temporal resolution. Directors can see the actors performance in real time, and watch the results on the mocap driven CG character. The unique marker IDs reduce the turnaround, by eliminating marker swapping and providing much cleaner data than other technologies. LEDs with onboard processing and a radio synchronization allow motion capture outdoors in direct sunlight, while capturing at 480 frames per second due to a high speed electronic shutter. Computer processing of modulated IDs allows less hand cleanup or filtered results for lower operational costs. This higher accuracy and resolution requires more processing than passive technologies, but the additional processing is done at the camera to improve resolution via a subpixel or centroid processing, providing both high resolution and high speed. These motion capture systems are typically under $50,000 for an eight camera, 12 megapixel spatial resolution 480 hertz system with one actor.

A high-resolution active marker system with 3,600 × 3,600 resolution at 480 hertz providing real time submillimeter positions.

Optical: Semi-passive Imperceptible Marker [edit]

One can reverse the traditional approach based on high speed cameras. Systems such as Prakash use inexpensive multi-LED high speed projectors. The specially built multi-LED IR projectors optically encode the space. Instead of retro-reflective or active light emitting diode (LED) markers, the system uses photosensitive marker tags to decode the optical signals. By attaching tags with photo sensors to scene



Summary, discussion and quiz

Summary, discussion and quiz

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

„ Image Processing and Computer Graphics"

Prezentacja multimedialna współfinansowana przez Unię Europejską w ramach Europejskiego Funduszu Społecznego w projekcie pt.

„Innowacyjna dydaktyka bez ograniczeń - zintegrowany rozwój Politechniki Łódzkiej - zarządzanie Uczelnią, nowoczesna oferta edukacyjna i wzmacniania zdolności do zatrudniania osób niepełnosprawnych"

Politechnika Łódzka

Politechnika Łódzka, ul. Żeromskiego 116, 90-924 Łódź, tel. (042) 631 28 83
www.kapitalludzki.p.lodz.pl