SCHOOL *of* ELECTRICAL ENGINEERING *and* COMPUTING
FACULTY *of* ENGINEERING *and* BUILT ENVIRONMENT
The UNIVERSITY *of* NEWCASTLE

Comp3320/6370 Computer Graphics

Course Coordinator: Associate Professor Stephan Chalup

Semester 2, 2018

LECTURE w01

Introduction to Computer Graphics

July 30, 2018

OVERVIEW

Class Times

- Main lecture: Mondays 10:00-12:00 in ES209
- Computer lab: Tuesdays 11:00-13:00 in EFG14

Course Objectives

- *Understand and appreciate computer graphics concepts*: We will provide an overview of computer graphics techniques with focus on techniques relevant for developing games and visualisations.
- *Appreciate mathematical fundamentals of computer graphics techniques*: The aim is to learn how to think geometrically using geometric and intuitive concepts that are timeless and have wide impact in computer graphics. We will also use algebraic concepts at a point where they can clarify the situation and make life of the graphics programmer so much easier.
- *Employ selected software to achieve interactive computer graphics*: The labs we will look at tools and tutorials for OpenGL.
- *Work within a group to complete a graphics project*: You will work like in a small game company. The project is very flexible so that your team can make some choices.
- *Be able to implement graphics effects*: As part of programming project you will work through some technical aspects in detail.

Prerequisites and Workload

- SENG1120 and MATH1110
- Vector geometry and linear algebra: dot product, cross product, linear independence, basis, cartesian coordinates, determinants, matrices
- Fundamental calculus and algebra: functions, differentiation, chain rule, curves, surfaces, real numbers, complex numbers, groups
- Programming: good C++ , C#, or Java skills
- Workload:
  - If you satisfy all prerequisites then about 10 hours per week (i.e. about 2 hours intense study per day).
  - Otherwise more study is required on top of that.
- Attending the lectures is highly recommended. Then you can directly interact with the lecturer in class and this makes study more efficient.

Literature

- (Akenine-Möller and Haines, 2002)
- (Akenine-Möller et al., 2008)
- (Shirley, 2009)
- (Angel and Shreiner, 2012)
- (Shreiner et al., 2006)
- (Shreiner et al., 2013)
- (Vince, 2010)
- (Hefferon, 2008)

Introduction

- *Computer Graphics* is about any use of the computer to generate or manipulate digital images.
- An *application program interface (API)* is a software interface that takes care of basic operations such as line drawing or drawing an image into a window. Examples are OpenGL, Direct3D, or Java3D.

Significance of Computer Graphics

The impact of computer graphics on many aspects of our life is growing rapidly.

"A picture says more than a thousand words."

Humans are Visual Beings

Why we like Computer Graphics so much:

- The visual cortex represents the largest and most sophisticated part of our sensory system.
- Humans think in images. These are primarily visual but can have other dimensions (eg. auditory, olfactory).
- Computer graphics can extend our visual abilities.
- Humans like pictures and graphics and understand them in most situations (elementary tasks) better and faster than text.
- Computer graphics allows us to communicate or interact with the computer more efficiently and in a more human–like manner.

Our representation or picture of our world is geometric:

- Scientists (eg. Albert Einstein) used geometry to analyse describe and understand our world.
- Engineers and architects draw graphical plans and diagrams to plan their systems (UML, CAD).
- Graphical user interfaces, 'Windows', facilitate work with computers.

Applications of Computer Graphics

If you are skilled in computer graphics you have a powerful tool which is essential for many tasks and jobs:

- GUIs
- CAD, GIS, Cartography
- Data analysis and visualisation
- Medical Imaging
- Simulations
- Computer arts and design
- Video or computer games
- Movies, effects
- Virtual reality, augmented reality
- Robotics, drones, autonomous cars

Future Development of Computer Graphics

- Computer hardware just became affordable and fast enough to produce really sophisticated graphics.
- There is a lot of computer software for computer graphics, geometry, visualisation and graphical data analysis (eg. Geomview, mathematica, matlab, ...).
- Computer graphics methods and hardware are now often used for other applications such as CUDA implementations in data mining or geometrical transformations in computer vision and robotics.

Computer graphics is not only a field for experts or people who can afford to buy expensive hardware. Computer graphics will boom and will be applied in many jobs and situations.

Skills for Computer Graphics

To be able to use computer graphics you need to acquire several skills and learn to apply them together:

- Mathematics and Geometry: numbers, vector spaces, matrices, linear algebra curves and surfaces, manifolds, topology, basic differential geometry
- Perspectives, projections, colours, photography
- Algorithms of computer graphics
- Programming skills, graphics libraries, OpenGL, DirectX
- Hardware, graphics cards, visual system
- Design, creativity

In our course we will start building up our skills from bottom up. We will in particular focus on aspects which are fundamental and (almost or less) timeless.

Why we need geometry in computer graphics

- Basic *rendering primitives* used by most graphics hardware: points, lines, triangles.
- A *model* or an *object* is a collection of geometric entities.
- A *scene* is a collection of models comprising that everything that is included in the environment is rendered. It can include material descriptions, lighting, and viewing specifications.
- Examples of objects: A car, a building, a line. Objects can consist of other objects.
- Dynamics and moving objects around: We need to understand how to map or transform an object via rotations, translations, scaling or shearing. This requires that we know mathematically how to do calculations with vectors and how to do mappings of points.

Geometrical Background

- In Computer Graphics we typically work with objects that live in a three-dimensional (Euclidean) world. The aim is to draw them in two dimensions on the screen.
- Some elementary maths can help us to make some of the calculations required in computer graphics simpler, clearer and much faster.
- Linear algebra covers properties of vectors and matrices required for displaying and modifying 3D objects in 3D space.
- Basic differential geometry helps to deal with non-linear curves and surfaces.

Notation

$\mathbb{R}$, $\mathbb{R}^+$, $\mathbb{R}^n$	real numbers, reals greater than 0, n-tuples of reals
$\mathbb{N}$	natural numbers: $\{0, 1, 2, ...\}$
$\mathbb{Z}$	integer numbers: $\{..., -1, 0, 1, 2, ...\}$
$\mathbb{Q}$	rational numbers, i.e fractions
$\mathbb{C}$	complex numbers
$\mathbb{H}$	Hamiltonian quaternions
$\{...; ...\}$	set of ... such that ...
$(a\,, b)$	open interval of reals between a and b
$[a\,, b]$	closed interval of reals between a and b
$(a\,, b]$, $[a\,, b)$	half-open intervals of reals between a and b
V, W, U	vector spaces
$\vec{v}, \vec{w}$	vectors
$\vec{v} \cdot \vec{w} = \langle \vec{v}, \vec{w} \rangle$	dot product of two vectors
$\vec{0}$, $\vec{0}_V$	zero vector, zero vector of V

Some Typical Maths Topics In Computer Graphics

Some topics which can be relevant for Computer Graphics

- Vector spaces, vectors, points
- Points, lines, planes, 3-space, ...
- Systems of linear equations
- Dot product, cross product
- Curves, surfaces, manifolds
- Matrices, transformations
- Triangulations, splines
- Complex numbers, quaternions
- More ..

Some Basic Algebraic Concepts

Please try to look up the definitions of the mathematical structures below in an Algebra book in the library or on-line (e.g. Bourbaki (1943) or Lang (2002)).

- sets
- monoids and semi-groups
- groups
- rings
- fields
- division rings, division algebras
- modules
- vector spaces

DEF. An *abelian group* is a set, V, together with an operation "$\cdot$" that combines any two elements a and b to form another element denoted $a \cdot b$. The following axioms have to be satisfied:

Closure For all $\vec{u}, \vec{v} \in V$ we have $\vec{u} \cdot \vec{v} \in V$.

Associativity For all $\vec{u}, \vec{v}, \vec{w} \in V$ we have $(\vec{u} \cdot \vec{v}) \cdot \vec{w} = \vec{u} \cdot (\vec{v} \cdot \vec{w})$.

Identity element There exists a identity element $Id \in V$ such that $\vec{v} \cdot Id = Id \cdot \vec{v} = \vec{v}$ for all $\vec{v} \in V$

Inverse element For each $\vec{v} \in A$ there exists an *inverse* $\vec{w} \in V$ such that $\vec{w} \cdot \vec{v} = \vec{v} \cdot \vec{w} = Id$.

Commutativity For all $\vec{u}, \vec{v} \in A$ we have $\vec{v} \cdot \vec{w} = \vec{w} \cdot \vec{v}$

Note: The operation "$\cdot$" could be replaced by "$+$" or anything that satisfies the axioms above.

DEF. By a *vector space over the real numbers* $\mathbb{R}$ we shall mean a set V along with two operations '$+$' and '$\cdot$' such that if $\vec{v}, \vec{w} \in V$ then $\vec{v} + \vec{w} \in V$ and if $r \in \mathbb{R}$ and $\vec{v} \in V$ then each *scalar multiple* $r \cdot \vec{v}$ of $\vec{v}$ is in V and the following five properties are satisfied:

VS 1. For all $\vec{u}, \vec{v}, \vec{w} \in V$ we have associativity, namely $(\vec{u} + \vec{v}) + \vec{w} = \vec{u} + (\vec{v} + \vec{w})$.

VS 2. There exists a *zero vector* $\vec{0} \in V$ such that $\vec{v} + \vec{0} = \vec{0} + \vec{v} = \vec{v}$ for all $\vec{v} \in V$

VS 3. For each $\vec{v} \in V$ there exists an *additive inverse* $\vec{w} \in V$ such that $\vec{w} + \vec{v} = \vec{v} + \vec{w} = \vec{0}$.

VS 4. For all $\vec{u}, \vec{v} \in V$ we have $\vec{v} + \vec{w} = \vec{w} + \vec{v}$

VS 5. If $r, s \in \mathbb{R}$ are *scalars* and $\vec{v}, \vec{w} \in V$ then $1 \cdot \vec{v} = \vec{v}$ and

- $(r + s) \cdot \vec{v} = r \cdot \vec{v} + s \cdot \vec{v}$
- $r \cdot (\vec{v} + \vec{w}) = r \cdot \vec{v} + r \cdot \vec{w}$
- $(rs) \cdot \vec{v} = r \cdot (s \cdot \vec{v})$

Notes about Vector Spaces

- **VS 1–4** say that V is an abelian group under vector addition.
- The additive inverse $\vec{w}$ in **VS 3** is uniquely determined and usually denoted by $-\vec{v}$.
- For all $r \in \mathbb{R}$ and for all $\vec{v} \in V$

$$r \cdot \vec{0} = 0 \cdot \vec{v} = \vec{0} \quad \text{and} \quad -(r \cdot \vec{v}) = (-r) \cdot \vec{v} = r \cdot (-\vec{v})$$

Terminology

- The elements of a vector space V are called *vectors*.
- To determine if a set V is a vector space one must specify the set V, and define vector addition and scalar multiplication in V. Then if V satisfies the properties **VS 1–5** it is a vector space over the real numbers $\mathbb{R}$.
- A vector space over the set of real numbers $\mathbb{R}$, is called a *real vector space*.
- Instead of using real numbers $\mathbb{R}$ any field can be employed such as the complex numbers $\mathbb{C}$. A vector space over the set of complex numbers $\mathbb{C}$, is called a *complex vector space*.

Examples

1. Let $V = \mathbb{R}^n$ be the set of n-tuples of real numbers. If $\vec{x} = (x_1, x_2, ..., x_n)$ and $\vec{y} = (y_1, y_2, ..., y_n)$ are two such n-tuples, define

$$\vec{x} + \vec{y} = (x_1 + y_1,\ x_2 + y_2, ...,\ x_n + y_n)$$

   and if $a \in \mathbb{R}$, define

$$a \cdot \vec{x} = (ax_1,\ ax_2, ...,\ ax_n)$$

   (Verify the vectorspace axioms.)

2. Let V be the set of all real valued functions on a non-empty set S. If f, g are functions, we can define $f + g$ in the usual way, and af in the usual way.

3. Let $V = M(n \times n, \mathbb{R})$ be the set of all $n \times n$–Matrices with real elements. Then V together with component-wise addition and component-wise multiplication with a scalar $a \in \mathbb{R}$ is a vector space over $\mathbb{R}$.

Subspaces

Given a vector space V over $\mathbb{R}$, a *subspace* of V is any nonempty subset W of V with

1. $\vec{0} \in W$ and
2. W is closed under addition (i.e., if $x, y, \in W$ then $x + y \in W$) and
3. W is closed under scalar multiplication (i.e., if $x \in W$ and $a \in \mathbb{R}$ the $ax \in W$)

It is easy to see that subspaces of V are vector spaces (over the same field) in their own right.

Example subspace

1. Let V be a vector space over $\mathbb{R}$, and let $v_1, v_2, ..., v_r \in V$. The subset of all *linear combinations*

$$x_1 v_1 + ... + x_r v_r, \quad \text{with } x_i \in R$$

   is a subspace (verify !). It is called the *subspace generated by* $v_1, v_2, ..., v_r$.

2. Let $n > 1$ and let j be a fixed integer, $1 \leq j \leq n$. Let $V = \mathbb{R}^n$ and let W be the set of all n-tuples in $\mathbb{R}^n$ whose j-th component is zero, i. e. $x_j = 0$. Then W is a subspace which is sometimes identified with $\mathbb{R}^{n-1}$ because it essentially consists of $(n-1)$-tuples.

DEF. A *norm* on a vector space V over the real numbers $\mathbb{R}$ is a function $\|.\| : V \longrightarrow \mathbb{R}, \vec{v} \mapsto \|\vec{v}\|$ such that following three properties are satisfied:

N1. For all $\vec{v} \in V \quad \|\vec{v}\| = 0 \Leftrightarrow \vec{v} = 0$.

N2. For all $\vec{v} \in V, \lambda \in \mathbb{R} \quad \|\lambda \cdot \vec{v}\| = |\lambda| \|\vec{v}\|$.

N3. For all $\vec{v}, \vec{w} \in V \quad \|\vec{v} + \vec{w}\| \leq \|\vec{v}\| + \|\vec{w}\|$ (triangle inequality).

Examples

- $\|\vec{v}\|_1 = \Sigma_{i=1}^{n} |v_i|$ (Manhattan norm or 1-norm)
- $\|\vec{v}\|_2 = \sqrt{\Sigma_{i=1}^{n} v_i^2}$ (Euclidean norm or 2-norm)
- $\|\vec{v}\|_p = (\Sigma_{i=1}^{n} |v_i|^p)^{\frac{1}{p}}$, where $p \geq 1$ is a real number (p-norm)
- $\|\vec{v}\|_\infty = max(|v_1|, ..., |v_n|)$ (maximum norm)

Note. $\|\frac{\vec{v}}{\|\vec{v}\|}\| = 1$

Basis of a vector space

DEF. The intersection of all subspaces of a vector space V containing a given set of vectors is called their *span*. The set of vectors is called *linearly independent* if no vector can be removed without diminishing the span. A linearly independent set whose span is the whole vector space V is called a *basis* of V.

Note. It can be shown that every vector space has a basis, and vector spaces over a given field are fixed up to isomorphism by a single cardinal number (called the *dimension of the vector space*) representing the size of the basis. For instance the real vector spaces are just $\mathbb{R}^0$, $\mathbb{R}^1$, $\mathbb{R}^2$, $\mathbb{R}^3$, ..., $\mathbb{R}^\infty$.

Linear Maps

DEF. Let V and W be two $\mathbb{R}$–vector spaces. A *linear map* from V to W is a map $V \longrightarrow W$ which preserves the vector space structure, i.e. it preserves sums of vectors and products of scalars with vectors.

The set of all linear maps from V to W is denoted $L(V, W)$. It is a $\mathbb{R}$–vector space. Given the bases of both vector spaces V and W, elements of $L(V, W)$ can be identified with matrices.

An *isomorphism* is a linear map that is one-to-one (*monomorphism*) and onto (*epimorphism*). Two vector spaces V and W are called *isomorphic* if and only if there exists an isomorphism between them.

Equations for Lines in 2D-Space [with (x,y)-coordinates]

- Basic form: $y = mx + b$ ($m =$ slope, $b =$ section on y-axis)
- Point & direction form: $y - p_2 = m(x - p_1)$
- Parametric vector form: $(x(t), y(t)) = p + tv = \begin{bmatrix} p_1 + tv_1 \\ p_2 + tv_2 \end{bmatrix}$
  (through point $p = (p_1, p_2)$ in direction of vector $v = (v_1, v_2)$)
- Two-Point form: $\frac{y-p_2}{q_2-p_2} = \frac{x-p_1}{q_1-p_1}$, through points p and q.
- Implicit form: $ax + by + c = 0$ (a, b, c are real constants).
- Axis section form: $\frac{x}{p_x} + \frac{y}{q_y} = 1$
  (line through points $p = (p_x, 0)$ and $q = (0, q_y)$)
- Hessian Normal Form (HNF): $x \cos\alpha + y \sin\alpha - d = 0$
  ($d =$ closest distance from origin, $\alpha =$ angle between normal and x-axis)

Some literature and sources

This lecture uses a number of texts as sources, including:

Angel and Shreiner (2012), Hefferon (2008), Shirley (2009), and Vince (2010).

The remaining slides and pictures of lecture 1 are mostly based on the book:
Tomas Akenine-Möller and Eric Haines and Natty Hoffman, Real-Time Rendering, 3rd Edition, 2008, isbn 987-1-56881-424-7, A. K. Peters, Ltd., Natick, MA, USA. Akenine-Möller and Haines (2002) or Akenine-Möller et al. (2008).
For this book there is an excellent support page at: http://www.realtimerendering.com/

Exercise 1

Calculate the midpoint of the circle in 2D that contains the following three points:

$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$B = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 5 \\ 1 \end{pmatrix}$$

Exercise 2

Show that the following two definitions of the dot product are equivalent (we restrict it to 3D now but is works the same in n dimensions):

$$\begin{aligned} \text{Def 1:} \quad & \vec{v} \cdot \vec{w} = v_1 w_1 + v_2 w_2 + v_3 w_3 \\ \text{Def 2:} \quad & \vec{v} \cdot \vec{w} = |\vec{v}|\,|\vec{w}| \cos\theta, \end{aligned}$$

where θ is the angle between $\vec{v}$ and $\vec{w}$.

Exercise 3

Select a plane and a line in 3D and calculate the "hitpoint":
For the plane you require a normal vector $\vec{n}$ and an anchor point B.
For the line you require a direction vector $\vec{c}$ and an anchor point A.
Use the example: $\vec{n} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$, $B = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix}$, $\vec{c} = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}$, $A = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$.

The Dot Product

The following two definitions of the dot product are equivalent (Exercise 2):

$$\begin{aligned} \text{Def 1:} \quad \vec{v} \cdot \vec{w} &= v_1 w_1 + v_2 w_2 + v_3 w_3 \\ \text{Def 2:} \quad \vec{v} \cdot \vec{w} &= |\vec{v}|\,|\vec{w}| \cos\theta \end{aligned}$$

The equivalence can be shown using the law of cosines applied to vectors $\vec{v}$ and $\vec{w}$:

$$\begin{aligned} |\vec{v} - \vec{w}|^2 &= |\vec{v}|^2 + |\vec{w}|^2 - 2\,|\vec{v}|\,|\vec{w}| \cos\theta \\ \text{i.e.} & \\ (v_1 - w_1)^2 + (v_2 - w_2)^2 + (v_3 - w_3)^2 &= v_1^2 + v_2^2 + v_3^2 + w_1^2 + w_2^2 + w_3^2 \\ & \quad -2\,|\vec{v}|\,|\vec{w}| \cos\theta \\ -2\,(v_1 w_1 + v_2 w_2 + v_3 w_3) &= -2\,|\vec{v}|\,|\vec{w}| \cos\theta \\ \text{Hence, } v_1 w_1 + v_2 w_2 + v_3 w_3 &= |\vec{v}|\,|\vec{w}| \cos\theta \end{aligned}$$

Geometrical Definitions

- Basic *rendering primitives* used by most graphics hardware: points, lines, triangles.
- A *model* or an *object* is a collection of geometric entities.
- A *scene* is a collection of models comprising that everything that is included in the environment is rendered. It can include material descriptions, lighting, and viewing specifications.
- Examples of objects: A car, a building, a line. Objects can consist of other objects.

The Computer Graphics Rendering Pipeline

- Main function of the pipeline: Given a virtual camera, 3D objects, light sources, lighting models, textures and other features generate, or *render*, a 2D image.
- Locations and shapes of objects in the image depend on: their geometry, placement of camera, and characteristics of environment.
- Appearance of objects depends on: material properties, light sources, textures, and lighting models.
- The pipeline has different stages. Some of them are implemented in hardware.
- Only primitives inside the *view frustum* are rendered.

Virtual Camera and View Frustum

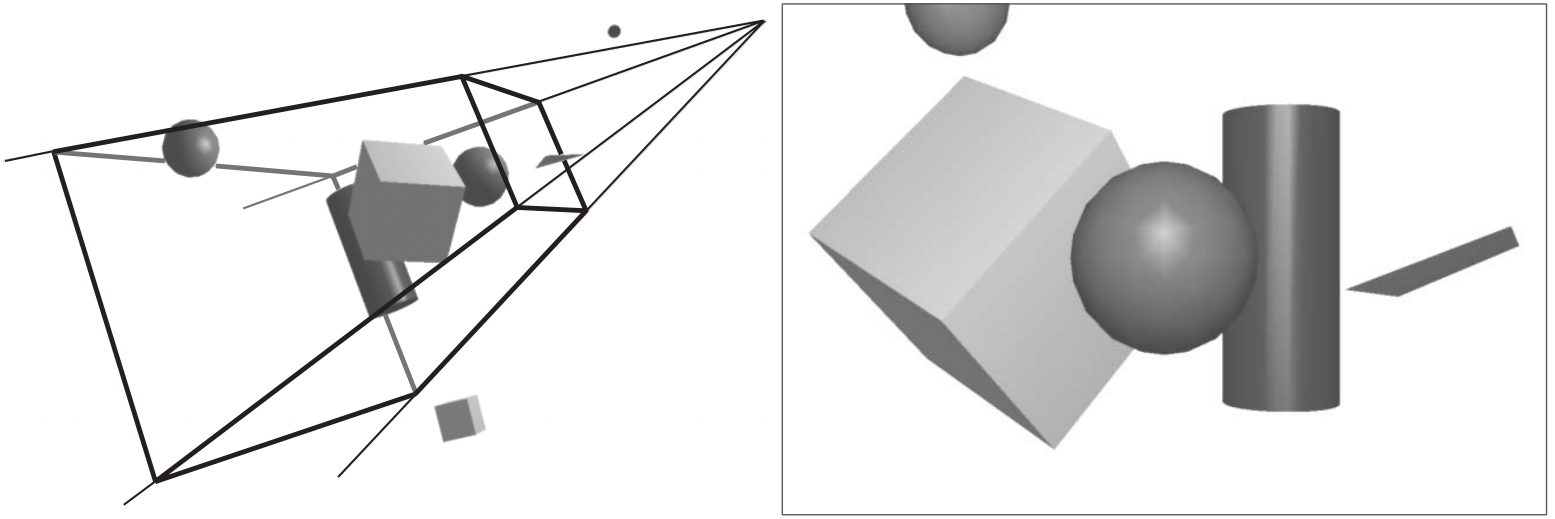


Figure 2.1. In the left image, a virtual camera is located at the tip of the pyramid (where four lines converge). Only the primitives inside the view volume are rendered. For an image that is rendered in perspective (as is the case here), the view volume is a frustum, i.e., a truncated pyramid with a rectangular base. The right image shows what the camera "sees." Note the bottommost cube in the left image is not in the rendering to the right because it is located outside the view frustum. Also, the triangle in the left image is clipped against the smaller (*near*) plane of the frustum, which results in a quadrilateral.

The Three Conceptual Stages

There are three conceptual stages:

1. *Application stage:* driven by the application and therefore implemented in software; it may contain collision detection, animations, acceleration algorithms
2. *Geometry stage:* implemented in soft- or hardware; transforms, projections, lighting, etc.
3. *Rasteriser stage:* draws an image using the data provided by the previous stage

Each of these stages typically is a pipeline in itself, i.e. it consists of several substages.

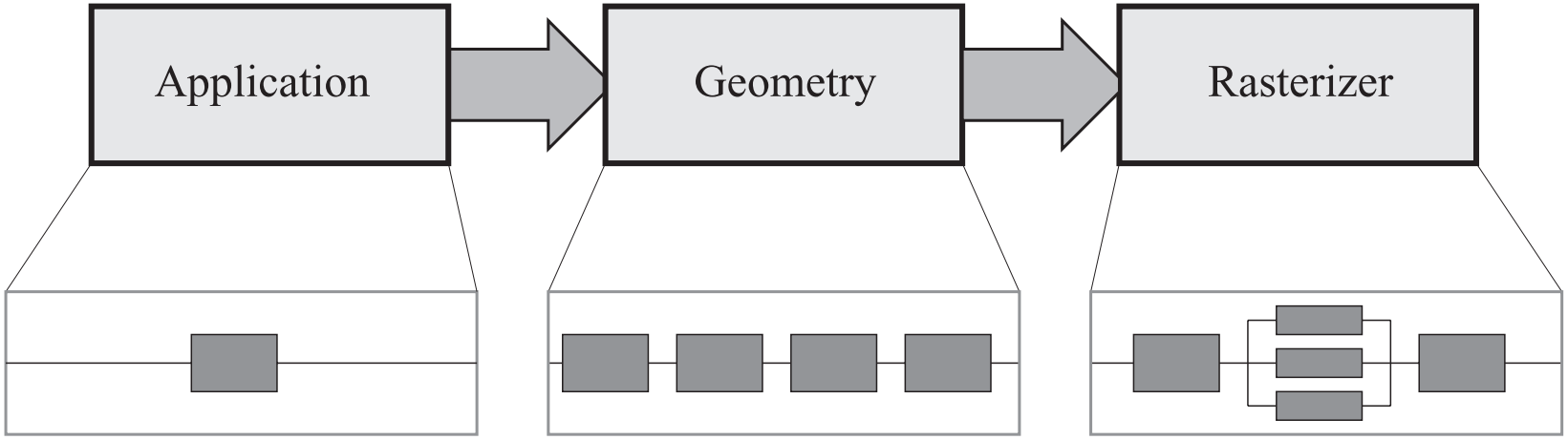


Figure 2.2. The basic construction of the rendering pipeline, consisting of three stages: application, geometry, and the rasterizer. Each of these stages may be a pipeline in itself, as illustrated below the geometry stage, or a stage may be (partly) parallelized as shown below the rasterizer stage. In this illustration, the application stage is a single process, but this stage could also be pipelined or parallelized.

The Application Stage

- It is software based and the developer has full control (The other stages are built upon hardware).
- Takes care of inputs from mouse, joystick keyboard, touchscreen, VR helmet, ...
- At the end of the application stage, the geometry (points, lines, triangles) is fed into the next stage.
- Typical processes of the application stage:
  - Collision detection $\rightsquigarrow$ response is fed back into colliding objects
  - Texture animation, animations via transforms, geometry morphing
  - Acceleration algorithms (e.g. hierarchical view frustum culling)

The Geometry Stage

It is subdivided in five functional stages:

Figure 2.3. The geometry stage subdivided into a pipeline of functional stages.

The geometry stage performs a very demanding task. With a single light source each vertex it requires about 100 individual precision floating point operations.

Model and View Transform

Model transform: applied to vertices and normals of the model for positioning and orienting.
View transform: places camera at origin and aims it in direction of neg. z-axis.

	model transform (4×4)-matrix		view transform (4×4)-matrix	
model space	$\rightsquigarrow$	world space	$\rightsquigarrow$	camera space
model coordinates model		world coordinates several instances		
		camera		neg. z–axis

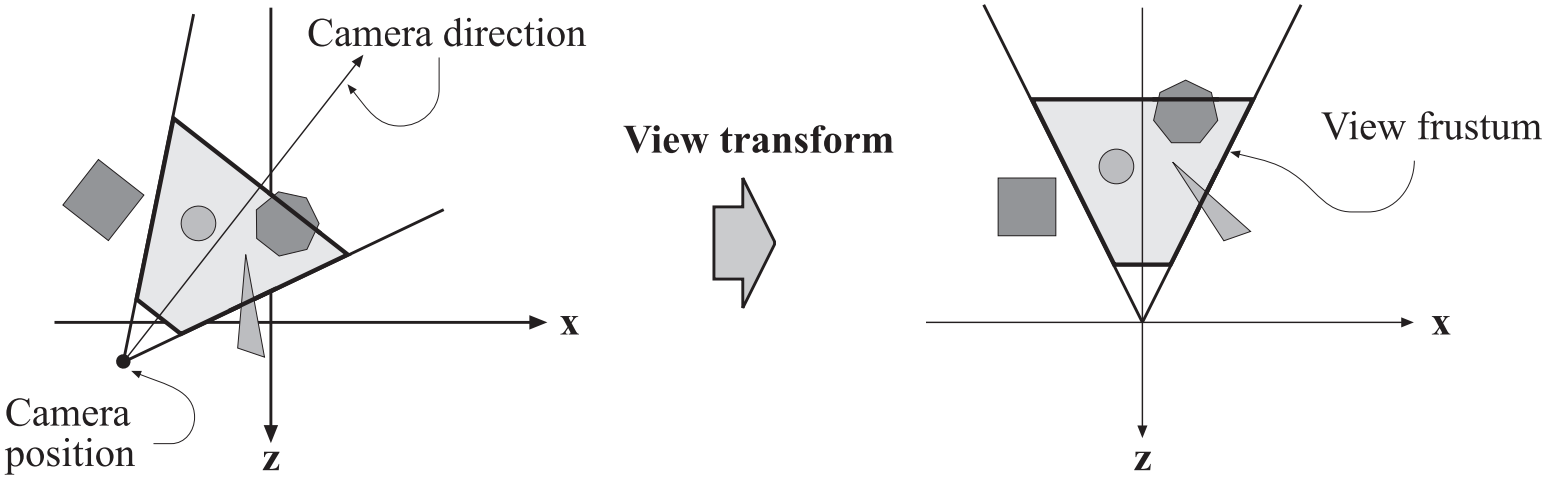


Figure 2.4. In the left illustration, the camera is located and oriented as the user wants it to be. The view transform relocates the camera at the origin looking along the negative z-axis, as shown on the right. This is done to make the clipping and projection operations simpler and faster. The light gray area is the view volume. Here, perspective viewing is assumed, since the view volume is a frustum. Similar techniques apply to any kind of projection.

Lighting and Shading

- One or more light sources.
- The geometric model may have a colour associated with each vertex.
- Texture can give a 3D effect without lighting.
- *Lighting equation:* Calculate colour for each vertex of the model.
- *Gouraud shading:* Use location and properties of light sources, position and normal of the vertex and material properties to compute the colour at each vertex of the surface. Then *interpolate* the colours at the vertex of each triangle over the triangle.
- *Pixel shading* can be used for more elaborate lighting effects.
- Normally, lighting is calculated in world space.

Lighting

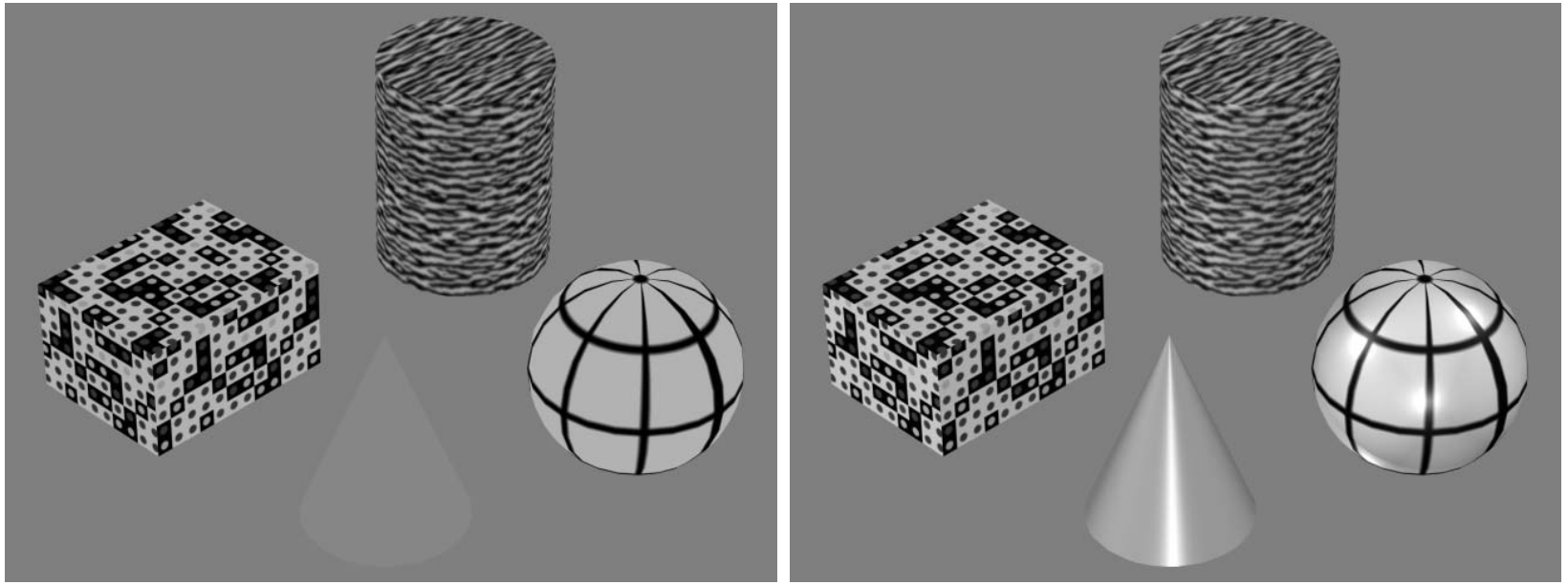


Figure 2.5. A scene without lighting is on the left; a lit scene on the right.

Projection

- After lighting the view volume is transformed into a unit cube with extreme points at (1,1,1) and (-1,-1,-1), called the canonical volume.
- Two projection methods
  1. Orthographic: parallel lines remain parallel; combine translation and scaling
  2. Perspective: The farther away the smaller; parallel lines may converge at horizon; mimics how we perceive object size.
- View frustum = truncated pyramide with rectangular base; is transformed into the unit cube.
- Both projections can be constructed with (4×4)-matrices.
- After projection the model is called to be in *normalised device coordinates*.
- z-coordinate is stored in the Z-buffer.

Orthographic and Perspective Projections

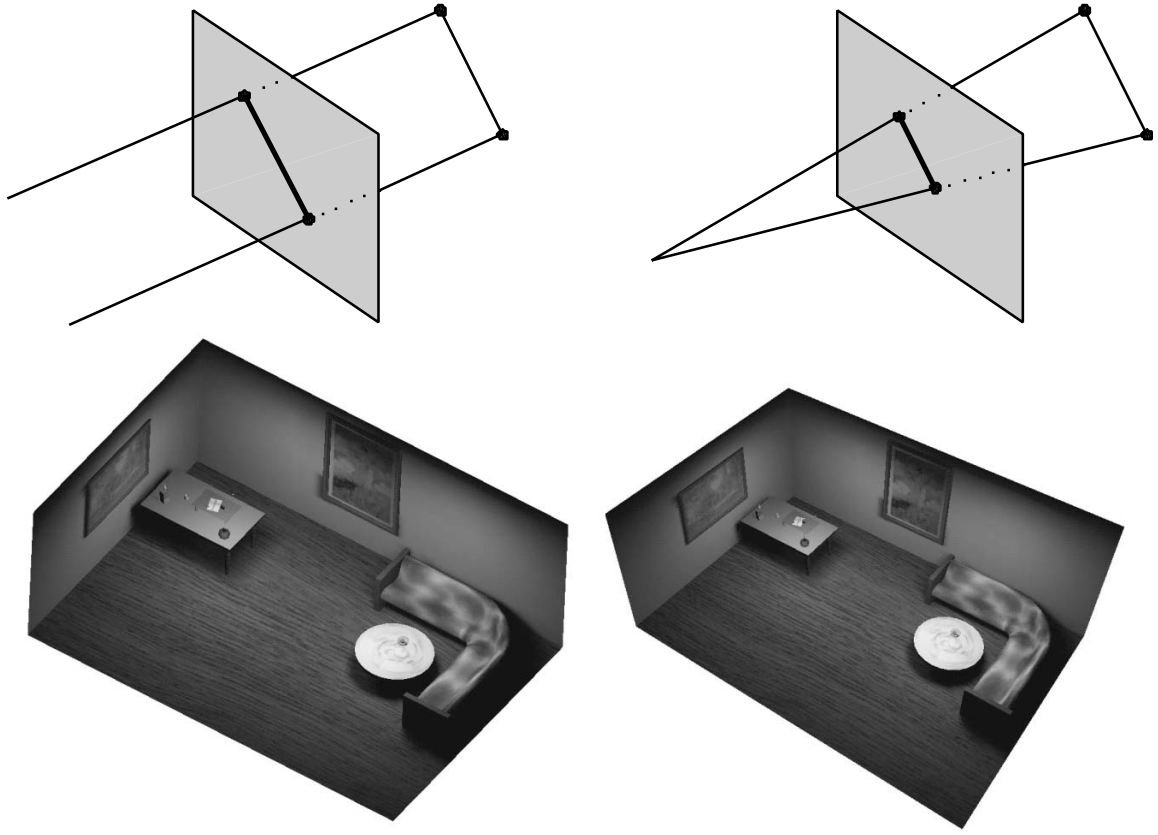


Figure 2.6. On the left is an orthographic, or parallel, projection; on the right is a perspective projection.

Clipping

- Only primitives wholly or partially inside the view volume need to be passed to the rasterizer stage, which then draws them on the screen.
- Due to the projection transform the transformed primitives are always clipped against the unit cube.
- The view volume has six clipping planes, but additional ones can be defined.

Clipping

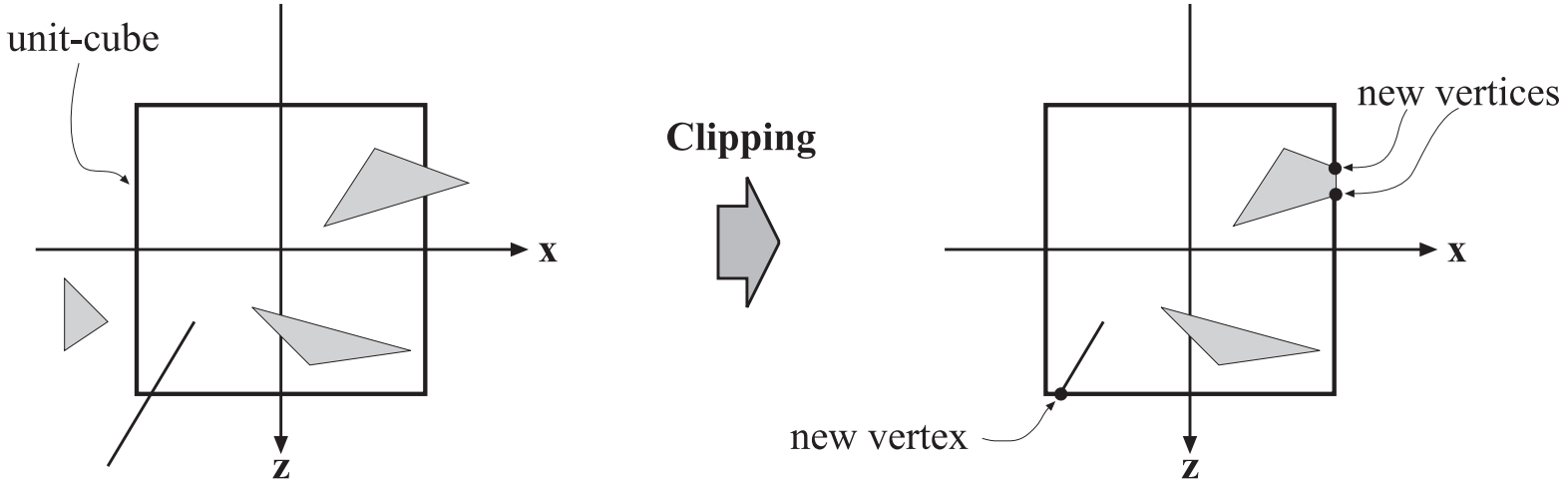


Figure 2.7. After the projection transform, only the primitives inside the unit cube (which correspond to primitives inside the view frustum) are desired for continued processing. Therefore, the primitives outside the unit cube are discarded, primitives totally inside are kept, and primitives intersecting with the unit cube are clipped against the unit cube, and thus new vertices are generated and old ones are discarded.

Screen Mapping

- Only the clipped primitives are passed on to the screen mapping stage.
- Still 3D.
- x-, and y-coordinates of each primitive are transformed to form screen coordinates.
- Translation and scaling.
- z-coordinate not affected ; $z \in$[-1,1].
- Screen coordinates together with the z-coordinate are passed on to the rasteriser stage.

Screen Mapping Process

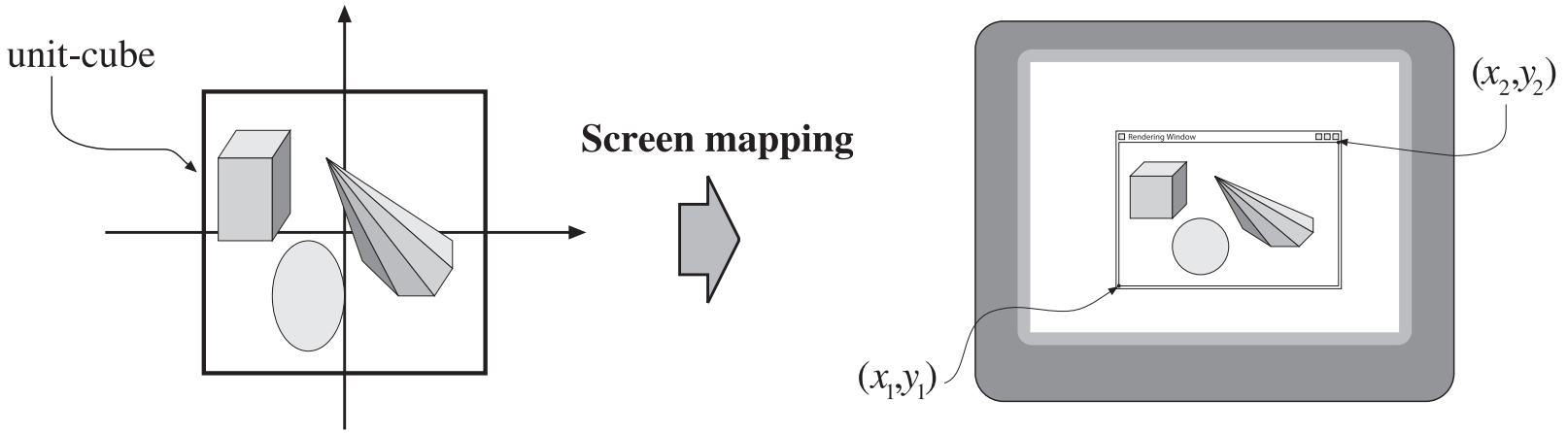


Figure 2.8. The primitives lie in the unit cube after the projection transform, and the screen mapping procedure takes care of finding the coordinates on the screen.

Summary of the Geometry Stage

In summary the geometry stage performs a very demanding task. With a single light source each vertex requires about 100 individual precision floating point operations.

Figure 2.3. The geometry stage subdivided into a pipeline of functional stages.

The Rasteriser Stage

- *Rasterization or Scan Conversion:* Given the transformed and projected vertices, colors, texture coordinates, assign correct colors to the pixels.
- Unlike the geometry stage which handles per polygon operations, the rasterizer stage handles per-pixel operations.
- Pixel information is stored in the *colour buffer* which is a rectangular array of colours.
- For high performance the rasterizer stage should be implemented in hardware.

Buffers, Visibility

- Double buffering uses a front and a back buffer to avoid that the viewer observes the rastering.
- Z-buffer $=$ depth buffer. It stores the z-value from the camera to the closest primitive.
- Z-buffer algorithm: If the polygon point being scan converted at (x, y) is no farther from the viewer than is the point whose colour and depth are currently in the buffers, then the new point's colour and depth replace the old values.
  - has $O(n)$ convergence
  - allows primitives to be rendered in any order
  - works for any primitive for which a z-value can be computed for each pixel
- Texturing means glueing an image to an object.

Texturing

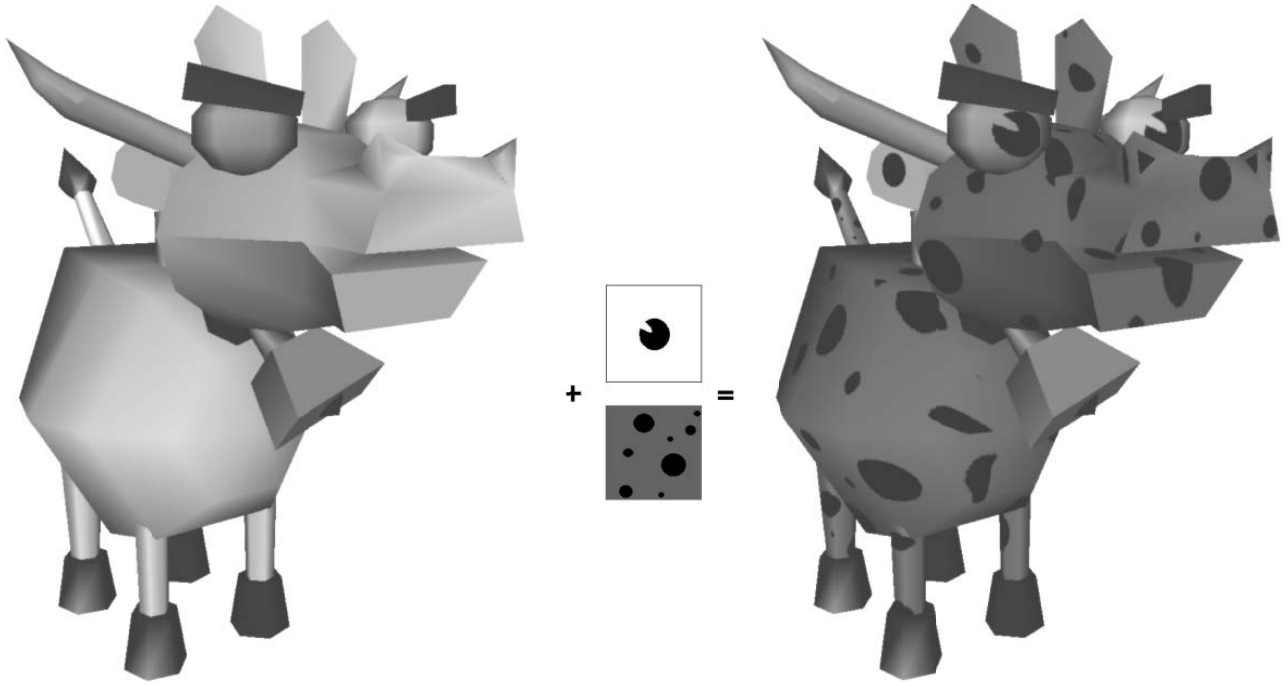


Figure 2.9. A cow model without textures is shown on the left. The two textures in the middle are "glued" onto the cow, and the result is shown on the right. The top texture is for the eyes, while the bottom texture is for the body of the cow. *(Cow model reused courtesy of Jens Larsson.)*

Summary of the Pipeline

- A model is built from points, lines and triangles; texture; light source.
- Application:
  - User can interact with the model.
  - Camera can move.
  - view transform and model transform.
- Geometry:
  - Put models into eye space.
  - Lighting, textures
  - Projection; unit cube; clipping; map into a window on the screen.
- Rasterizer:
  - All primitives are rasterized.
  - Render primitives with texture.
  - Visibility: Z-buffer algorithm.

Exercise 4

Given a vector $\vec{v}$ in 3D, how can we obtain a perpendicular vector (i.e. a vector that is orthogonal to $\vec{v}$)?
How can we test that it is perpendicular?
Is it uniquely determined?

Exercise 5

Given vectors $\vec{v}$ and $\vec{w}$ in 3D. How can we obtain the projection of $\vec{v}$ to $\vec{w}$?

Exercise 6

Given vectors $\vec{v} = (2, 2, 1)^T$ and $\vec{w} = (1, -2, 0)^T$ calculate:
a) The dot product $\vec{v} \cdot \vec{w}$
b) The cross product $\vec{v} \times \vec{w}$
c) The projection of $\vec{v}$ onto $\vec{w}$, i.e. the projection vector $proj_{\vec{w}}(\vec{v})$ and its length $\|proj_{\vec{w}}(\vec{v})\|$.
(Note: We use the notation $proj_{\vec{w}}(\vec{v})$ and not $proj_{\vec{v}}(\vec{w})$.)

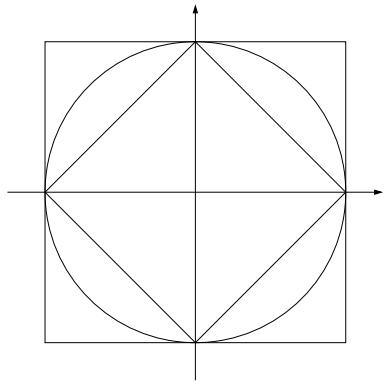
Exercise 7

The unit ball in n-dimensional space $\mathbb{R}^n$ is the set of points

$$B^n = \{\vec{v}; \vec{v} \in \mathbb{R}^n \text{ and } \|\vec{v}\| \leq 1\}$$

Describe how the unit ball in $n = 2$ dimensions looks like for:
a) The 1-norm $\|\vec{v}\|_1 = \Sigma_{i=1}^{n} |v_i|$
b) The 2-norm $\|\vec{v}\|_2 = (\Sigma_{i=1}^{n} |v_i|^2)^{\frac{1}{2}}$, i.e. the Euclidean norm.
c) The maximum norm $\|\vec{v}\|_\infty = \max_{i=1,\ldots,n} |v_i|$

Solution hint: Which is which?



Exercise 8 (Matrix product)

Let $M \in M(m \times k, \mathbb{R})$ and $N \in M(k \times n, \mathbb{R})$ be two matrices with real coefficients. Let $P \in M(m \times n, \mathbb{R})$ be the matrix obtained by multiplying M and N:

$$P = (p_{ij})_{\substack{i=1,\dots,m \\ j=1,\dots,n}} = M \cdot N$$

Give the general formula for p_{ij}, i.e. the coefficients of P.

LITERATURE

Akenine-Möller, T. and Haines, E. (2002). *Real-Time Rendering*. A K Peters, second edition.

Akenine-Möller, T., Haines, E., and Hoffman, N. (2008). *Real-Time Rendering*. A K Peters, third edition.

Angel, E. and Shreiner, D. (2012). *Interactive Computer Graphics, A Top-Down Approach with Shader-Based Open GL*. Addison-Wesley, sixth edition.

Bourbaki, N. (1943). *Elements of Mathematics, Algebra I, Chapters 1–3*. Hermann, Publishers in Arts and Science, 293 rue Lecourbe, 75015 Paris, France.

Hefferon, J. (2008). *Linear Algebra*. Freely available on the web.

Lang, S. (2002). *Algebra*. Springer, New York, 3rd edition.

Shirley, P. (2009). *Fundamentals of Computer Graphics*. A K Peters, third edition.

Shreiner, D., Sellers, G., Kessenich, J., and Licea-Kane, B. (2013). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3*. Addison Wesley, eighth edition.

Shreiner, D., Woo, M., Neider, J., and Davis, T. (2006). *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison Wesley, fifth edition. “The red book".

Vince, J. (2010). *Mathematics for Computer Graphics*. Springer-Verlag.