2018

Paper Three Computer Science Notes

UNIVERSITY OF CAMBRIDGE, PART IA

ASHWIN AHUJA

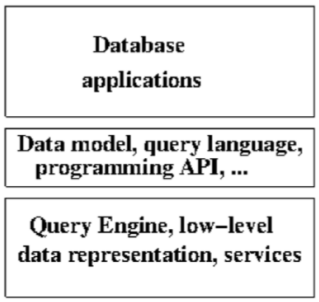
Table of Contents

Databases

Introduction

Abstraction of data management: In an ideal world, you should be able to alter the implementation of how the data which is stored is to be used through the idea of abstraction. How the data stored is a form of abstraction that we should be able to ignore when writing the use of the data.

Database Management System (DBMS): A query engine knows about the low-level details hidden by the interface, using this knowledge to optimise the query evaluation. A DBMS first and foremost offers **persistent storage** while offering (1) CRUD operations and (2) ACID transactions.



N.B. For the purposes of the course, the application APIs and the networking APIs will be ignored.

CRUD:

1. Create
2. Read
3. Update
4. Delete

ACID

1. Atomicity
    a. Either all actions of a transaction are carried out, or none are (even if the system crashes in the middle of a transaction).
2. Consistency
    a. Every transaction applied to a consistent database leaves it in a consistent state.
3. Isolated
    a. Transactions are isolated from the effects of other concurrently executed transactions.
4. Durability
    a. If a transaction completes successfully, then its effects persist.

Different Data Models

1. Relational Model: data is stored in tables (SQL is the main query language).
2. Graph-oriented Model: Data is stored as a graph (nodes and edges), **The Query languages tend to have path-oriented features.**
3. Aggregate-oriented Model: Optimised for reads – they are also called document oriented database.

Why multiple options for DBMS:

1. No one system nicely solves all data problems
2. Several fundamental trade-offs faced by application designers and system designers.
    a. One important trade-off involves redundant data
    b. **Data in a database is redundant if it can be deleted and then reconstructed from the data remaining in the database.**
3. Database engine might be optimised for a particular class of queries
4. Query language might be tailored to the same class.

Data Redundancy (Query Response vs Update Throughout):

Data redundancy is problematic for some applications – if a database supports many concurrent updates, then data redundancy leads to many problems. If a database has little redundancy, then update throughout is typically better since transactions need to only lock a few data items.

However, in a low redundancy database, evaluation of complex queries can be very slow and require large amounts of computing. By precomputing, answering common queries (either fully or partially) can greatly speed up the queries. This introduces redundancy, but it may be appropriate for databases that are read-intensive, with few data modifications.

Source of data for the course (IMDB):

- Data taken is a subset of the information available from IMDb – taken from the top 10 movies from 2006 through 2017.

Relational Databases

ER Diagrams

Implementation Independent technique to describe the data that we store in a database. ER was devised by Peter Chen. The technique grew up around relational databases systems but it can help document and clarify design issues for any data model.

- **Entities (Represented using a Square):** represents the items of our model
- **Attributes (Ovals):** represents properties
- **Key (Underlined):** Key is an attribute whose value uniquely identifies an entity instance.
    - They are often automatically generated to be unique. They might be formed from some algorithm – it is generally a better idea to generate key rather than using things which are out of control. The only safe thing to use as a key is something that is automatically generated in the database and only has meaning within that database.
- The **scope** of the model is limited – among the vast number of possible attributes that could be associated with an **attribute, we are declaring that our model is concerned with a specific amount of them.**
- **Relationships (Diamonds):** represents the connections between entities – 'the verbs'
    - Relationships can have attributes indicating information about the relationship.

ER Diagrams are very abstract and are independent of implementation.

Weak Entities

Weak entities are entities which depend on the existence of another entity. The relationship which connects the two entity is known as the **identifying relationship.**

Cardinality of a Relationship

The relation *R* is

one-to-many: Every member of *T* is related to at most one member of *S*.

many-to-one: Every member of *S* is related to at most one member of *T*.

one-to-one: *R* is both many-to-one and one-to-many.

many-to-many: No constraint.

Relational DBMSs:

- In the 1970s, in writing a database application, you needed to know a lot about the data's low-level representation.
- Codd instated the system of giving the users a model of data and a language for manipulating that data which is independent of the details of the representation / implementation. The model is based on mathematical relations.

Mathematical Relations

Cartesian Product:

Suppose S and T are sets. The Cartesian product, S x T is:

$S \times T = \{(s,t) : s \in S, t \in T\}$

A (binary) relation over S x T is any set R with: $R \subseteq S \times T$

- S and T are referred to as domains.
- We are interested in finite relations R that can be stored.

n-ary relation R is a set:

$R \subseteq S_1 \times S_2 \times \ldots \times S_n = \{(s_1, s_2, \ldots, s_n) : s_i \in S_i\}$

Instead of a set of tuples, we use a table, with column names and record which is effectively a set of items consisting of a value for each column (attribute)

Mathematical vs Database Relation

Named Columns: A name (attribute name) is associated with each domain. Instead of tuples, records are used (set of pairs consisting an attribute name with a value in each domain)

Column Order: Does not matter for database relations. The **schema** of a database relation is defined as R(A1 : S1, A2 : S2, … An : Sn)

Relational Database Query Language

It effectively acts as a function which takes a collection of relation instances and returns a single relation instance. In order to meet Codd's goals, we want a query language that is high level and is independent of physical data representation. A couple of possibilities are relational algebra and SQL.

Relational Algebra and SQL

Relational Algebra Reference

$$
\begin{array}{rcll}
Q & ::= & R & \text{base relation} \\
 & | & \sigma_p(Q) & \text{selection} \\
 & | & \pi_{\mathbf{X}}(Q) & \text{projection} \\
 & | & Q \times Q & \text{product} \\
 & | & Q - Q & \text{difference} \\
 & | & Q \cup Q & \text{union} \\
 & | & Q \cap Q & \text{intersection} \\
 & | & \rho_M(Q) & \text{renaming}
\end{array}
$$

- p is a simple boolean predicate over attributes values.
- $\mathbf{X} = \{A_1, A_2, \ldots, A_k\}$ is a set of attributes.
- $M = \{A_1 \mapsto B_1, \ A_2 \mapsto B_2, \ \ldots, \ A_k \mapsto B_k\}$ is a renaming map.
- A query Q must be **well-formed**: all column names of result are distinct. So in $Q_1 \times Q_2$, the two sub-queries cannot share any column names while in in $Q_1 \cup Q_2$, the two sub-queries must share all column names.

SQL intro:

- Made up of multiple sub-languages, including Query Language, Data Definition Language and System Administration Language
- It is an evolving language and has changed consistently over multiple stages over standardization.
- (Has its origins at IBM in the early 1970s)

Selection

R

A	B	C	D
20	10	0	55
11	10	0	7
4	99	17	2
77	25	4	0

$\Longrightarrow$

$Q(R)$

A	B	C	D
20	10	0	55
77	25	4	0

Q

RA $\sigma_{A>12}(R)$

SQL
```
SELECT DISTINCT * FROM R WHERE R.A > 12
```

The use of the 'DISTINCT' keyword is important and the reason for it will be discussed later.

Projection

R

A	B	C	D
20	10	0	55
11	10	0	7
4	99	17	2
77	25	4	0

$\Longrightarrow$

$Q(R)$

B	C
10	0
99	17
25	4

Q

RA $\pi_{B,C}(R)$

SQL `SELECT DISTINCT B, C FROM R`

Renaming

R

A	B	C	D
20	10	0	55
11	10	0	7
4	99	17	2
77	25	4	0

$\Longrightarrow$

$Q(R)$

A	E	C	F
20	10	0	55
11	10	0	7
4	99	17	2
77	25	4	0

Q

RA $\rho_{\{B \mapsto E,\ D \mapsto F\}}(R)$

SQL `SELECT A, B AS E, C, D AS F FROM R`

Union

R

A	B
20	10
11	10
4	99

S

A	B
20	10
77	1000

$\Longrightarrow$

$Q(R, S)$

A	B
20	10
11	10
4	99
77	1000

Q

RA $R \cup S$

SQL `(SELECT * FROM R) UNION (SELECT * FROM S)`

Intersection

R

A	B
20	10
11	10
4	99

S

A	B
20	10
77	1000

$\Longrightarrow$ $Q(R)$

A	B
20	10

Q

RA $R \cap S$

SQL `(SELECT * FROM R) INTERSECT (SELECT * FROM S)`

Difference

R

A	B
20	10
11	10
4	99

S

A	B
20	10
77	1000

$\Longrightarrow$ $Q(R)$

A	B
11	10
4	99

Q

RA $R - S$

SQL `(SELECT * FROM R) EXCEPT (SELECT * FROM S)`

Product

R

A	B
20	10
11	10
4	99

S

C	D
14	99
77	100

$\Longrightarrow$ $Q(R, S)$

A	B	C	D
20	10	14	99
20	10	77	100
11	10	14	99
11	10	77	100
4	99	14	99
4	99	77	100

Q

RA $R \times S$

SQL `SELECT A, B, C, D FROM R CROSS JOIN S`

SQL `SELECT A, B, C, D FROM R, S`

Note that the RA product is not exactly the Cartesian product suggested by this notation!

Joins

Natural Join

Notation

- Often, the domain types are ignored and the relational schema is written as R(A) where A is a set of attribute names.
- When R(A, B) is written **you implicitly mean R(A U B).**
- You can do equality on an attribute, eg:
  - u.[A] = v.[A] is equivalent (abbreviates) u.A1 = v.A1 ^ u.A2 = v.A2 … u.An = v.An

Natural Join for R(a, b) and R(b, c) is defined as:

$$\boldsymbol{R} \bowtie \boldsymbol{S} = \{\boldsymbol{t} \mid \exists \boldsymbol{u} \in \boldsymbol{R}, \boldsymbol{v} \in \boldsymbol{S}, \boldsymbol{u}.[\boldsymbol{B}] = \boldsymbol{v}.[\boldsymbol{B}] \wedge \boldsymbol{t} = \boldsymbol{u}.[\boldsymbol{A}] \cup \boldsymbol{u}.[\boldsymbol{B}] \cup \boldsymbol{u}.[\boldsymbol{C}]\}$$

In relational algebra, it is written as:

$$R \bowtie S = \pi_{\mathbf{A},\mathbf{B},\mathbf{C}}(\sigma_{\mathbf{B}=\mathbf{B}'}(R \times \rho_{\vec{\mathbf{B}} \mapsto \vec{\mathbf{B}}'}(S)))$$

Implementation

Since the Entity Relationship does not dictate the implementation, there are a normally a number of options in implementing the model. Normally, each comes with its own drawbacks, so we must weigh them up.

For example, we could simply put all information into one table, however, this has a lot of problems with data redundancy since there is a lot of data which must appear in multiple places.

Anomalies caused by data redundancy:

1. Insertion Anomalies
   a. How can we tell if a newly inserted record is consistent with all other records for that thing? Also, we might want to insert a record without needing to give all the information, In the case of the movie table, one may want to define a table without defining the director of the movie.
2. Deletion Anomalies
   a. We will wipe out information about sub-entities when the last record which has this contained inside which we might not want to remove.
3. Update Anomalies
   a. All records for the same thing may differ due to things like typos, thus creating a problem.
4. **Concurrency Issues**
   a. **A transaction implementing a conceptually simple update but containing checks may end up locking the entire table.**

Therefore, it is much better to split the database up into a number of different tables each of which you work on.

Join in SQL

- Use a join … on … structure
- Eg:

```
select movies.id as mid, title, year,
       people.id as pid, name, gender
from movies
join directs on movie_id = movies.id
join people on people.id = person_id
```

- **COMPLEXITY**
  - In the worst case, using a method like this to do the join, the function is O(mn) where m and n are the number of elements in each previous table.

```
for each (a, b) in R {
   // scan S
   for each (b', c) in S {
      if b = b' then create (a, b, c) ...
   }
}
```

Observations

- Both ER entities and ER relationships are implemented as tables
- Good: We avoid many update anomalies
- Bad: We have to work hard to combine information in tables (joins) to produce effective queries.

Ensuring consistency of a Relational Database

In order to ensure that a table representing a relation actually implements a relation using keys and **foreign keys**

Keys

Superkeys

Relational Key

Suppose $R(\mathbf{X})$ is a relational schema with $\mathbf{Z} \subseteq \mathbf{X}$. If for any records u and v in any instance of R we have

$$u.[\mathbf{Z}] = v.[\mathbf{Z}] \Longrightarrow u.[\mathbf{X}] = v.[\mathbf{X}],$$

then $\mathbf{Z}$ is a superkey for R. If no proper subset of $\mathbf{Z}$ is a superkey, then $\mathbf{Z}$ is a key for R. We write $R(\underline{\mathbf{Z}}, \mathbf{Y})$ to indicate that $\mathbf{Z}$ is a key for $R(\mathbf{Z} \cup \mathbf{Y})$.

Foreign Keys and Referential Integrity

If we have a superkey or key from a different table in another table (normally representing a relationship), it is a foreign counter in that table. A database is said to have **referential integrity** when all foreign key constraints are satisfied.

Operation of Primary Keys and Foreign Keys

```
create table ActsIn (
   movie_id int not NULL,
   person_id int not NULL,
   character varchar(255),
   position  integer,

   primary key (movie_id, person_id),
   constraint actsin_movie
         foreign key (movie_id)
         references Movies(id),
   constraint actsin_person
         foreign key (person_id)
         references People(id))
```

primary key automatically contains the constraint of ensuring that it is unique.
The foreign key constraint effectively checks that It actually references an item in the defined table.

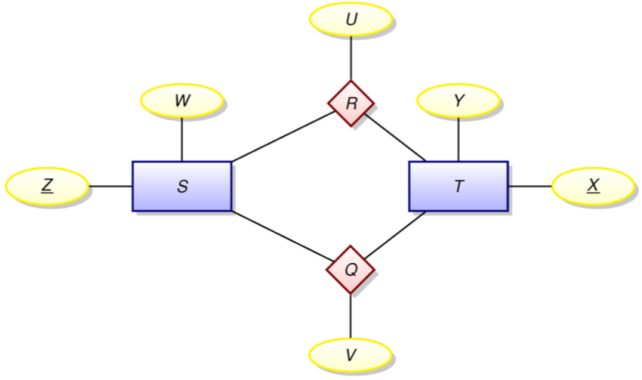
Further Implementation

Design of Relationships ('clean' approach)

Relation R is	Schema
many to many ($M:N$)	$R(\underline{X}, \underline{Z}, U)$
one to many ($1:M$)	$R(\underline{X}, Z, U)$
many to one ($M:1$)	$R(X, \underline{Z}, U)$
one to one ($1:1$)	$R(\underline{X}, Z, U)$ and/or $R(X, \underline{Z}, U)$

However, items can differ from the clean approach, for example the ability of NULLs means that you could entirely forego a relationship instead simply filling items which don't exist (didn't have a relationship) to have a NULL. This is generally how we would implement **weak entities (since that means it is definitely a 1-1 relationship).**

Implementing multiple relationships in a single table:



Rather than using two tables

$$R(\underline{X,\ Z},\ U)$$
$$Q(\underline{X,\ Z},\ V)$$

we might squash them into a single table

$$RQ(\underline{X,\ Z},\ type,\ U,\ V)$$

using a tag $domain(type) = \{\mathbf{r}, \mathbf{q}\}$ (for some constant values r and q).

- represent an R-record (x, z, u) as an RQ-record $(x, z, \mathbf{r}, u, NULL)$
- represent an Q-record (x, z, v) as an RQ-record $(x, z, \mathbf{q}, NULL, v)$

While this would work, it definitely introduces new redundancy as given the value of the type column, you can get the value of either the U or V column.

Database Index: An index is a data structure – created and maintained within a database system that can greatly reduce the time required to locate records. However, there are many types of database index (it is not defined in the SQL standards). It will also take more time to complete updates (even if it is faster to read).

A database index can be created and deleted thus:

```
CREATE INDEX index_name on S(B)

DROP INDEX index_name
```

Multisets

SQL in fact creates a multiset not a set (that is multiple rows can have the same values as this is very important as data can repeat). This means that if you want to only select the set equivalent (therefore removing the no distinct elements, you need to use the work 'distinct'.

Using Duplicates

Duplicates are important for aggregate functions (min, max, average, count) and can easily be used by utilising the 'GROUP BY' command. The Group By splits the thing into separate tables for each group before being reassembled when using an aggregate function.

3 Valued Logic

SQL has NULLs – it is a placeholder. It is not a member of any domain (therefore x <> 10 will return a 'false' for NULL). This also means we need three-values logic:

$\wedge$	**T**	**F**	$\perp$
T	**T**	**F**	$\perp$
F	**F**	**F**	**F**
$\perp$	$\perp$	**F**	$\perp$

$\vee$	**T**	**F**	$\perp$
T	**T**	**T**	**T**
F	**T**	**F**	$\perp$
$\perp$	**T**	$\perp$	$\perp$

v	$\neg v$
T	**F**
F	**T**
$\perp$	$\perp$

it is important to note that NULL is ambiguous:

1. Value exists but we don't know it
2. No value is applicable

3. The value is known, but you aren't allowed to see it.
4. **IS NULL can be used to imply NULL = NULL (GROUPING BY NULL) but NULL = NULL is NULL.**

Partial Functions as Relations

- A partial function $f \in S \to T$ can be thought of as a binary relation where $(s,t) \in f$ iff t = f(s)
- Suppose R is a relation where if (s,t1) is a member of R and (s, t2) is also, then it follows that t1 = t2. Here, R represents a (partial) function.
- Given partial functions $f \in S \to T \; and \; g \in T \to U \; then \; (f \circ g)(s) = g\big(f(s)\big) = g \circ f \in S \to U$
- Since $Q \circ R \equiv R \bowtie_{2=1} Q$ we can see that **joins are a generalisation of function composition.**

Views: By defining the table as a result of a query, this can now be used in later queries as if it was a table. Eg:

```
create view coactors as
  select DISTINCT c1.person_id as pid1,
                  c2.person_id as pid2
  from credits AS c1
  join credits AS c2 on  c2.movie_id = c1.movie_id
  where c1.type = 'actor'
    and c2.type = 'actor';
```

Now cofactors(pid1 pid2) can be used as if it is a table.

Graph Databases

Definitions

G = (V, A) is a directed graph where:

V is a finite set of vertices (nodes)

A is a binary relation over V **that is** $\boldsymbol{A \subseteq V \times V}$

If $(u,v) \in A$ then we also have an arc from u to v.
The arc (u, v) is also called a directed edge, or a relationship of u to v.

Composition: $A \circ A$ consists of the nodes which can be reached in a path of length 2

Iterated Composition (paths)

Iterated Composition is defined as R^1 = R and $R^{n+1} = R \circ R^n$
A path in a graph where v1, v2, … , vk is the sequence of vertices, we can write the path as v1 -> v2 -> v3 etc
N.B. If G = (V, A) is a directed graph, and (u, v) are in A^k , then there is at least one path in G from u to v of length k. Such paths may contain loops.

Shortest Path

The R-distance (hop count): Suppose s0 is a member of the vertices of R, such that there is a pair (s0, s1) which is a member of R:

- The distance from s0 to s0 is 0
- If (s0, s1) is a member of R then the distance from s0 to s1 is 1.
- For any other s' which is a member of the vertices of R, the distance from s0 to s' is the least n, such that (s0, s') is a member of R^n
- **THE BACON NUMBER IS THE R-distance where s0 is Kevin Bacon**

BACON NUMBER in SQL

```
create view bacon_number_1 as
 select distinct pid2 as pid, 1 as bn
 from coactors
 where pid1 = 121299 and not pid2 = 121299;

create view bacon_number_2 as
 select distinct pid2 as pid, 2 as bn
 from coactors
 join bacon_number_1 on pid1 = pid
 where pid2 not in (select pid from bacon_number_1)
   and not pid2 = 121299;

create view bacon_number_3 as
 select distinct pid2 as pid, 3 as bn
 from coactors
 join bacon_number_2 on pid1 = pid
 where pid2 not in (select pid from bacon_number_1)
   and pid2 not in (select pid from bacon_number_2)
   and not pid2 = 121299;

create view bacon_number_4 as
 select distinct pid2 as pid, 4 as bn
 from coactors
 join bacon_number_3 on pid1 = pid
 where pid2 not in (select pid from bacon_number_1)
   and pid2 not in (select pid from bacon_number_2)
   and pid2 not in (select pid from bacon_number_3)
   and not pid2 = 121299;
```

Transitive Closure

The transitive closure is the smallest binary such that the relation is a subset of the original relation and is transitive:

$$(x, y) \in R^+ \wedge (y, z) \in R^+ \rightarrow (x, z) \in R^+$$

Therefore, has all paths, hence can be expressed as:

$$R^+ = \bigcup_{n \in \{1,2,3,\dots\}} R^n$$

Since our relations are finite, there must be some k with:

$$R^+ = R \cup R^2 \cup \dots \cup R^k$$

However, k will depend on the contents of R! and hence we **cannot** compute transitive closure using relational algebra.

Hence, the idea comes to design and implement a database system that is **optimised for transitive close and related path-oriented queries.**

Neo4j

- Contains nodes and binary relationships between nodes.
  - The nodes and relationships can have attributes (called **properties**).
- Neo4j has a query language called Cypher that contains path-oriented constructs. It is designed to explore very large graphs.
- The general idea is you use pattern matching to match paths and return their values. An example of how to compute all Bacon Numbers is listed below but you **MUST** look at the introduction to Cypher and ticks to remember how to use Cypher.

Bacon Numbers

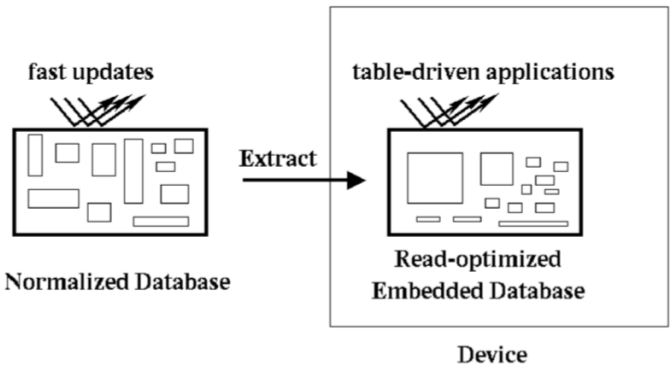
```
MATCH (p:Person)
where p.name <>  "Bacon, Kevin (I)"
with p
match paths=allshortestpaths(
      (m:Person {name : "Bacon, Kevin (I)"} )
         -[:ACTS_IN*]- (n:Person {name : p.name}))
return distinct p.name,
       length(paths)/2 as bacon_number
order by bacon_number desc;
```

Read-Optimised Databases

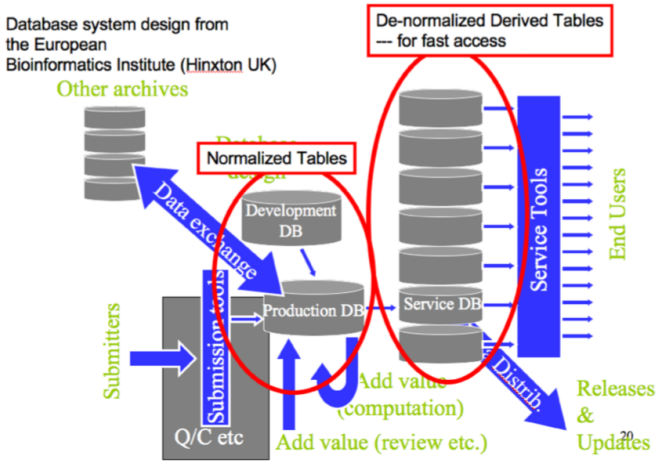
Fundamental Tradeoff: Introducing data redundancy can speed up read-oriented transactions at the expense of slowing down write-oriented transactions. Sometimes, however, the tradeoff is worth it, in order to have a fast reading capability:

1. If data is seldom updated, but read from very often.
2. Reads can afford to be out of sync with the write-oriented database. Then periodically extract read-oriented snapshots and storing them in a database optimised for reading.

(2)



This should be used in a fetch intensive data organisation, where information is required rapidly even if it is okay if the data is out of date. An example of where this is used is for Hinxton Bio-informatics, where there are normalized tables and de-normalised derived tables (which service the end users).



Semi-Structured Data – JSON

You know how it looks like, but it is separated by different tags with items (and the name of the columns stored) in quotation marks.

JSON Example

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

From `http://json.org/example.html`.

Semi-Structured Data – XML

The data is stored and separated using HTML like start and end tags.

XML Example

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

From `http://json.org/example.html`.

Document-oriented database systems
Stores data in the form of **semi-structured objects.** Such database systems are also called **aggregate-oriented databases.**

Items are semi-structured, in order to speed up the speed of retrieval of data from the database as well as reducing the amount of redundant data that needs to be stored. The use of semi-structured databases means that we are then able to find items using a key-value retrieval process.

Key-value stores
One of the simplest types of database systems is the **key-value store** that simply maps a key to a block of bytes. The retrieved block of bytes is typically opaque to the databases system. The interpretation of such data is left to applications.

This describes a **pure** key-value store. Some key-value stores extend this architecture with some capabilities to inspect blocks of data and extract meta-data such as indices – this is true with BerkleyDB (using the DoctorWho bespoke data store).

DoctorWho
DoctorWho is written in Java – having a lot of functions which are written for the purpose. An example of a piece of code is provided below but again, you should read the introduction which is provided. Though, this is very unlikely to be tested on, since you would need a reference sheet and the code is specific to this storage of data.

DoctorWho Code Example

```
import uk.ac.cam.cl.databases.moviedb.MovieDB;
import uk.ac.cam.cl.databases.moviedb.model.*;

public class GetMovieById {
    public static void main(String[] args) {
        try (MovieDB database = MovieDB.open(args[0]))
            int id = Integer.parseInt(args[1]);
            Movie movie = database.getMovieById(id);
            System.out.println(movie);
        }
    }
}
```

Application

Sarah Mei
Previously, a relational database was the only type of database that was every used. Today, however, there is a rising sentiment of a NoSQL policy, which suggests that SQL (and the relational model) is not always the most effective type of database. Sarah Mei talked in particular about how it is ineffective in particular for a Social Networking platform.

It is apparently clear that the relational model is bad for social data, so the site chose to use a document database. This is attractive as all the data is located where you need it. However, this is also dangerous, since updating a user's data means going though all the streams where the data appears and updating it there. This is **error prone** and often leads to inconsistent data and mysterious errors (especially with deletion).

She says that if there is no value in the links between documents – the data is not necessarily tied together, then a document database is effective and would work perfectly. However, definitely not good if there is value in the link – especially not in a social network where the links are the most vital part of the entire system.

OLAP vs OLTP

OLTP: **Online Transaction Processing**
OLAP: **Online Analytical Processing**

	OLAP	OLTP
Supports	**analysis**	**Day-to-day operations**
Data is	**Historical**	**Current**
Transactions mostly	**Reads**	**Updates**
Optimised for	**reads**	**Updates**
Data redundancy	**High**	**Low**
Database size	**humongous**	**Large**

OLAP
OLAP is commonly associated with decision support and data warehousing. It generally involves looking at historical data (which won't get updated anymore) and getting information from it which could be used in the future.

SQL is generally bad for people to understand – you want things like pivot tables and cross tabulation. Standard SQL does not handle these well. Therefore you want to consider other query languages or indeed a different database model – such as graph based databases.

Data Cube
In the data cube, data is modelled as a n-dimensional hypercube. Each dimension is associated with a column (a hierarchy). Each point on the cube records information and aggregation and cross-tabulation is possible along all dimensions.

You can roll up or drill down (going back) into any one dimension, in order to get more information about something in particular

Star Schema
In practise, tables can be very large with hundreds of columns. Row-oriented table stores can be very inefficient since a typical query is concerned with only a few columns and needs to retrieve data from every record.

Therefore, completing a column-oriented implementation is often much faster, though it is much harder to modify an item (therefore utilised for OLAP).

+ easy to add/modify a record

- might read in unnecessary data

+ only need to read in relevant data

- tuple writes require multiple accesses

=> suitable for read-mostly, read-intensive, large data repositories

Distributed Databases

Why Distribute Data?

1. Scalability
    a. **The dataset is too large for a single machine**
2. Fault Tolerance
    a. **In case one machine (or some) fail, the system can continue to function**
3. Lower Latency
    a. **Data can be located closer to all users, therefore meaning it would be much faster for them. Also, more servers to simply deal with requests will just mean that it would be faster for everyone.**

How distribute data?

1. **Replication**
    a. Information is simply copied between every server (this would not work where the information is too large to be stored)
2. **Partition**
    a. Information is split between systems – therefore would not be faster for users in specific geographic locations – but would be more scalable.
    b. Partitions themselves are often replicated.

In reality, you would generally attempt to do the second (with replication of the partitions as well, meaning all the advantages of data distribution are received.

CAP Concepts

These are the things you attempt to gain in a distributed system, but are often impossible.

1. **CONSISTENCY**
    a. All reads return data that is up-to date.
2. **AVAILABILITY**
    a. All clients can find some replica of the data.
3. **PARTITION TOLERANCE**
    a. The system continues to operate despite arbitrary message loss or failure of part of the system

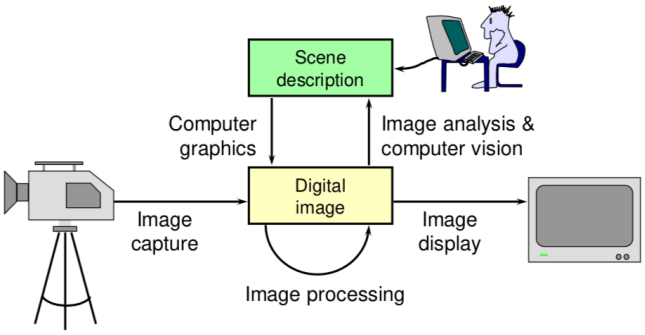
CAP Principle:

- Assume that network partitions and other connectivity problems will occur.
- Implementing transactional semantics is very difficult and slow
- It is always **a trade-off between availability and consistency**
    - **This gives rise to the notion of EVENTUAL CONSISTENCY:** if update activity ceases, then the system will eventually reach a consistent state.

Graphics

Background

Computer Graphics and image processing is the process of taking a scene description and converting it into some sort of image which can be processed.



It is very important, since all visual computer output (and every kind of visual imagery) depends on CG:

- Printed output
- Monitor
- TV, movie special effects and post-production
- Books, magazines, catalogues, brochures, junk mail, newspapers, packaging, posters and flyers.

Vision

There are three requirements for vision:

1. Illumination
2. Objects
3. Eyes

It should be noted that light that we see is only a small subset of the entire electromagnetic spectrum. This is the visible light spectrum, which is between 300nm and 800nm. (**The long wavelength is red and the short wavelength is violet**).

What is an image?

An image is a two dimensional (continuous function) f(x, y) - the value at any point is an intensity / colour. It is important to be noted that an image is not digital per se, the function is completely continuous.

Eye

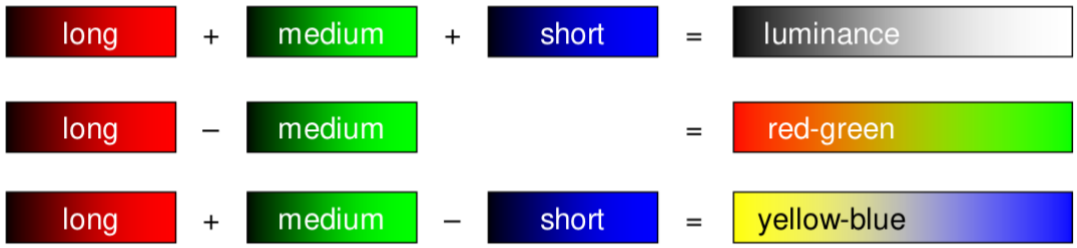
- **Structure of the eye**
  - The lens focuses light onto the retina (an array of light detection cells in the back of the eye).
    - The shape of the lens can change to change the distance on which we are focusing.
  - The **fovea** is the high-resolution area of the retina (has the greatest concentration of cones).

- The optic nerve takes signals from the retina to the visual cortex in the brain. It is here the pieces are pieced together and we can perceive an image.

- **Light Detectors**
  - There are two classes of detector: **rods** and **cones**
    - The cones come in three types: sensitive to short, medium and long wavelengths.
  - The cones are concentrated in the **macula**, at the centre of the eye
  - **The fovea is a densely packed region in the centre of the macula**: it contains the highest density of cones (therefore highest resolution vision)

Colour Signals

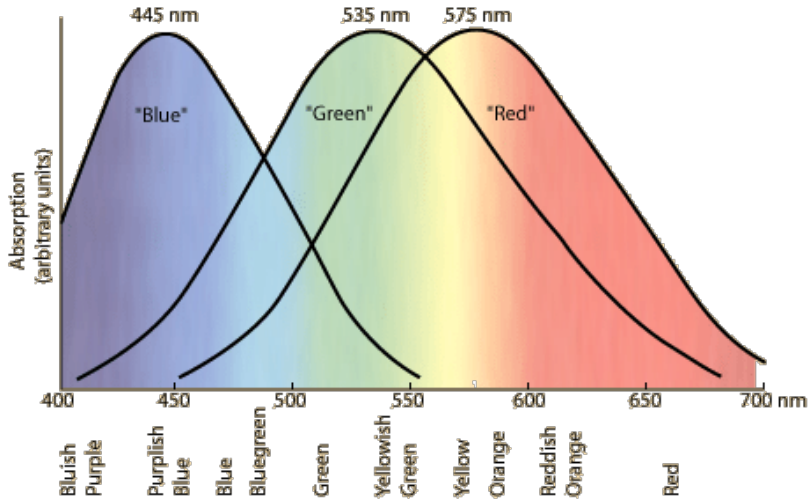
The signals which are sent to the brain are pre-processed by the retina:



It is easy to find the scale for which they correspond to by simply consider what would happen in the maximum and minimum.

These shows why red, green yellow and blue are perceptually important colours.

It is possible, by mixing different amounts of red, green and blue lights, to generate a wide range of responses in the human eye – but it must be noted that not all colours can be created this way.



Colour-Blindness Effects

Colour blindness can be caused by one of two methods:

1. The person is missing a cone (therefore has only got two cones).
2. The person has a shifted cone

a. This means that colours between the two are generally harder to differentiate, therefore may not be able to be differentiated.

Digital Image

This is a contradiction in terms: if you can see it, it's not digital. However, we define a digital image as a sampled and quantised version of a real image. It is a rectangular array of intensity or colour values.

Sampling

A digital image is a rectangular array of intensity values – each value is called a pixel (picture element). The sampling resolution is normally measured in pixels per inch (ppi). This is equivalent to the number of dots per inch (dpi).

- Computer Monitors – 100 ppi
- Lasers and Inkjet printers – 300 – 1200 ppi
- Typesetters – 1000 – 3000 ppi

Image Capture

- Lots of devices can be used:
    - Scanners
        - Line CCD (Charge Coupled Device) in a flatbed scanner
        - Spot detector in a drum scanner
    - Cameras
        - Area CCD
        - CMOS camera chips

Sampling resolution: The sampling resolution determines how many pixels the image should be made into. For each point, the colour of the actual image is averaged into the colour of the digital image which is stored.

Quantisation

Quantisation is the process of converting a continuous value of the intensity of a bit of the image into a digital number stored in a number of bits.
It limits the number of different intensities which can be stored (and limits the maximum brightness that can be stored).
For human consumption, generally only 8 bits are required (though some applications use 10, 12 or 16 – generally to be down-quantised to 8 bits).
The colour is stored normally as a number for each of the red, green and blue – with an intensity (8 bits) for each one.

Storing Images in Memory

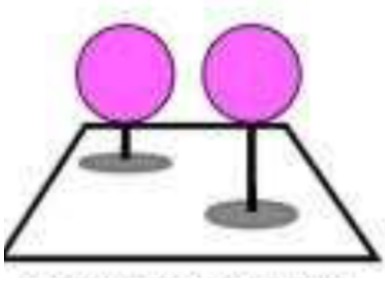
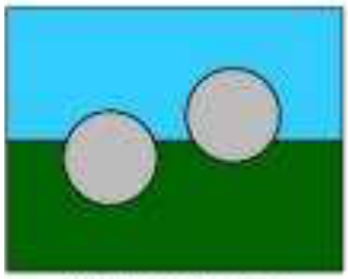
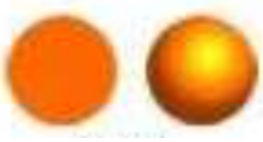
8 bits used to be the de facto standard for greyscale images – but 16 bits is now generally more used.
For 8 bits: pixel[x][y] is stored at base + x + WidthOfImage x y
This is necessary since the memory is one dimensional whereas the images are two dimensional.

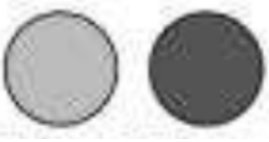
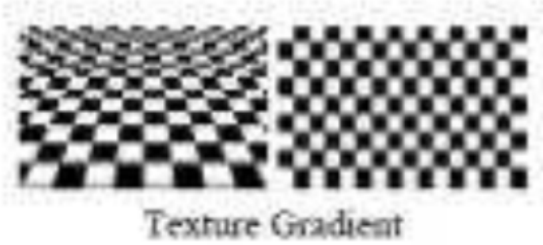
Frame Buffer

Computers have a special piece of memory reserved for storage of the current image being displayed – the frame buffer. This normally consists of **dual ported DRAM (Dynamic RAM)** – this is sometimes referred to as **Video RAM.**

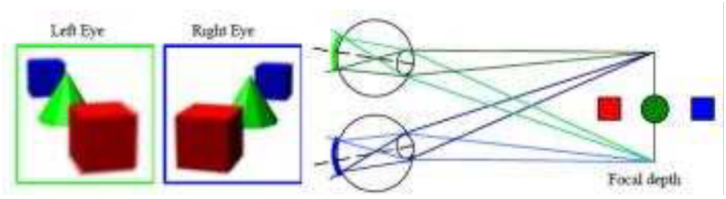
Rendering

Depth Cues

1. Occlusion
    a. The object in front covers the object behind.
    b. 
2. Relative Size
    a. Objects which are further away are smaller than objects nearby.
    b. 
3. Shadow and Foreshortening
    a. The shadow of an object on other objects and the floor as well the shortening of distances as they go further away (**Foreshortening**)
    b. 
4. Distance to horizon
    a. The closer the items are to the horizon, they are further they are away.
    b. 
5. Shading
    a. A 3D object would have shading to do with the texture and shape of the object – for example a sphere would have shadows towards the back of the object.
    b. 
6. Colour

   a. The colour of different objects may give us hints as to whether they are nearer or closer – this would be to do with the light that is near them – for example a blue light source at the back of a scene would make objects them bluer.
7. Relative Brightness
   a. For luminescent objects, depending on how bright they are, you can see how near they are to you – the brighter the closer they are. Also, depending on where the lights are this can also work for other objects. Eg. If the source is at the front of the scene objects the nearer to you would be brighter.
   b. 
8. Atmosphere
   a. If there is a (thick or thin) atmosphere, the nearer the object, the more in focus the object would be – the light would hit lest dissipate less.
   b. 
9. Focus
   a. The more in focus an object is, the closer it is likely to be. You can also tell more practically based on the focal length of the item used to photograph the thing.
   b. 
10. Familiar Size
   a. You can use items of a known size in order to gain an understanding of where the object is.
   b. In case of repeated objects, this would also be possible.
   c. 
11. Texture gradient
   a. A texture gradient (which continues backwards) which can be used to gain an understanding for where objects are.
   b. 

12. 3D images (left and right eye)
   a. In 3D images (or just real-life objects), different images are shown to the eye – therefore when we look at an object (and focus on it), we can see where it is, relative to other objects.

b. 

When we render depth, we can use all of these techniques to ensure that it can be seen where objects are relative to each other when they are displayed as two-dimensional objects.

Drawing Perspective

Throughout history, the perspective tended to be very wrong as artists failed to properly make things appear to be three dimensional.

Examples:

1. Early – Ambrogio Lorenzetti 1332 – Presentation at the Temple
2. Wrong Early – Lorenzo Monaco 1407 – 1409 – Adoring Saints
3. Renaissance Perspective – Filippo Brunelleschi 1413 – Geometrical Perspective (Holy Trinity fresco)
4. More (better) – Carlo Crivelli 1486 – The Annunciation of Saint Emidius

They were also fascinated by the idea of false perspective – the idea that 'impossible' designs could be drawn by warping perspective; Also, by using clever steps and changing the size of objects, you could make someone make someone look much larger than the other person.

Calculating Perspective: In the old days, the perspective would be found (when drawing an object) by using a piece of string from behind the painter going to the point and finding the point it went through the canvas and marking it for a number of points.
Then these could be joined together to make an object in perspective.
This method is known as ALBERTI'S ONE POINT PERSPECTIVE

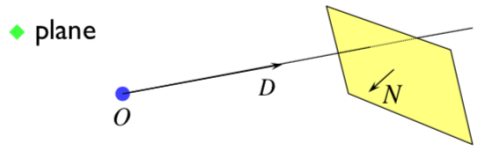
Ray Tracing

- The method involves finding the point on the surface and calculating the illumination.
- Given a set of 3D objects, you shoot a ray from the eye through the centre of every pixel and see what surface it hits – this defines the colour of the pixel.
- It easily **handles reflection, refraction, shadows and blur**
- **But it is very computationally expensive.**

```
select an eye point and a screen plane

FOR every pixel in the screen plane
    determine the ray from the eye through the pixel's centre
    FOR each object in the scene
        IF the object is intersected by the ray
            IF the intersection is the closest (so far) to the eye
                record intersection point and object
            END IF ;
        END IF ;
    END FOR ;
    set pixel's colour to that of the object at the closest intersection point
END FOR ;
```

Calculating intersections of ray with object

Plane:



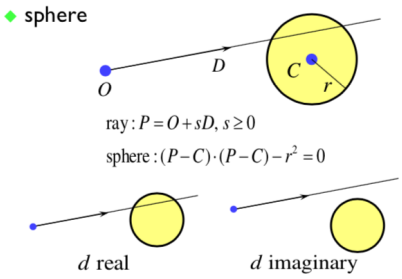
ray: $P = O + sD,\ s \geq 0$

plane: $P \cdot N + d = 0$

$$s = -\frac{d + N \cdot O}{N \cdot D}$$

Polygon or Disc: Intersection of the ray with the plane of the polygon and then check whether the intersection lies within the polygon.

Sphere:



$$a = D \cdot D$$
$$b = 2D \cdot (O - C)$$
$$c = (O - C) \cdot (O - C) - r^2$$
$$d = \sqrt{b^2 - 4ac}$$
$$s_1 = \frac{-b + d}{2a}$$
$$s_2 = \frac{-b - d}{2a}$$

Cylinder:

Deal with it is as two discs as the top and bottom discs then deal with the finite curved surface of the cylinder:

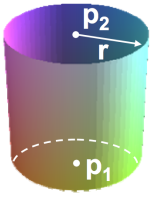
The finite cylinder (without caps) is described by equations:

$(q - p_a - (v_a, q - p_a)v_a)^2 - r^2 = 0$ and $(v_a, q - p_1) > 0$ and $(v_a, q - p_2) < 0$

The equations for caps are:

$(v_a, q - p_1) = 0, (q - p_1)^2 < r^2$ bottom cap

$(v_a, q - p_2) = 0, (q - p_2)^2 < r^2$ top cap



Torus

The easiest way I've found to deal with a torus is by converting the ray into Cartesian form and find the intersection using the general formula of the torus:

$$\left(x^2 + y^2 + z^2 + R^2 - r^2\right)^2 = 4R^2\left(x^2 + y^2\right).$$

If there is no solution then they do not intersect.

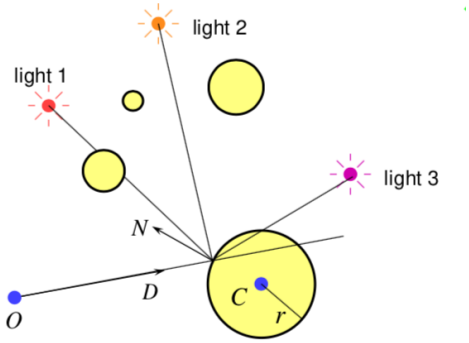
Shading

Once you have the intersection of a ray with the nearest object you can:

- Calculate the normal to the object at the intersection point.
- Shoot rays from that point to all the light sources and calculate the diffuse and specular reflections off the object at that point. **This (plus ambient illumination) gives you the colour of the object (at that point),**

Shadows

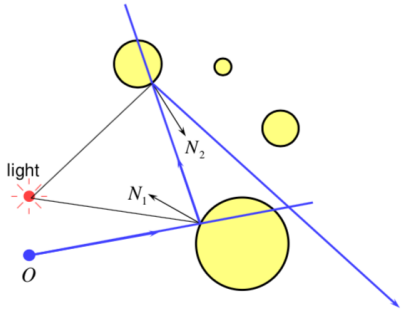
When going for the raw from the intersection point to the light, you can check whether another object is between the intersection and the light and is hence casting a shadow. You also need to watch for self-shadowing.



\

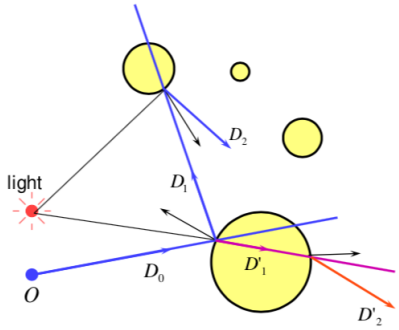
Reflection

If a surface is totally or partially reflective then new rays can be spawned to find the contribution to the pixel's colour given by the reflection.



Transparency and Refraction

Objects can be totally or partially transparent – allowing objects behind the current one to be seen through it. The transparent objects can have refractive indices (bending the rays as they pass through the object). This means the ray can be split into two parts.



Illumination and Shading

Durer's method allows us to calculate what part of the scene is visible in any pixel (using a string to go from the eye to the object).

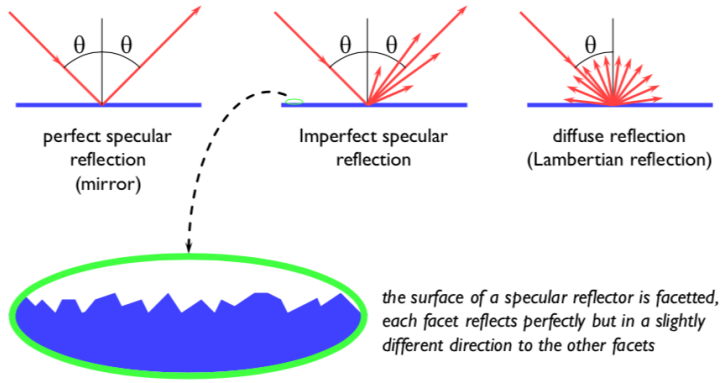
The colour of the pixel depends on:

- Lighting
- Shadows

- Properties of Surface Material

Reflection of Light

Light can be considered to reflect in a few ways – through perfect specular reflection, imperfect specular reflection (still concentrated about perfect reflection) and diffuse reflection (where the light is completely even going everywhere).



Also, the surface can absorb some wavelengths of light, for different types of reflection.

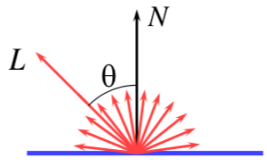
Plastics are good examples of surfaces with **specular reflation in the light's colour** and **diffuse reflection in the plastic's colour.**

In calculating the shading of a surface, we make a few assumptions:

1. There is diffuse reflection **and approximate specular reflection**
2. All light falling on a surface comes directly from a light source
    a. There is no interaction between objects
    b. **However, we can cheat and say that all light reflected off all other surfaces onto a given surface can be amalgamated into a single specular term: ambient illumination and add this to the diffuse and specular reflection.**
3. No objects cast shadows on any other
    a. So, we can treat each surface as if it were the only object in the scene.
4. Light sources are considered to be infinitely distance from the object (lights at infinity)
    a. So, the vector to the light is the same across the whole surface.

We can observe that the colour of a flat surface will be uniform across it, dependent on the colour and position of the object and the colour and position of the light source.

Diffuse Shading



$$I = I_l k_d \cos\theta$$
$$= I_l k_d (N \cdot L)$$

L is a normalised vector pointing in the direction of the light source

N is the normal to the surface

I_l is the intensity of the light source

k_d is the proportion of light which is diffusely reflected by the surface

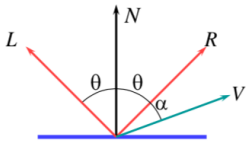
I is the intensity of the light reflected by the surface

- We can have different Ii and kd for different wavelengths (colours)

- If cos(theta) is less than 0 the light is behind the polygon – so does illuminate this side of the polygon.
- One-sided vs two-sided surfaces
    - **One sided:** Only the side in the direction of the normal vector can be illuminated – therefore if cos(theta) < 0 then both sides are black
    - **Two sided:** The sign of cos(theta) defines which side of the polygon is illuminated.

Specular Reflection

✦ Phong developed an easy-to-calculate *approximation* to specular reflection



$$I = I_l k_s \cos^n \alpha$$
$$= I_l k_s (R \cdot V)^n$$

Phong Bui-Tuong, "Illumination for computer generated pictures", *CACM*, 18(6), 1975, 311–7

L is a normalised vector pointing in the direction of the light source

R is the vector of perfect reflection

N is the normal to the surface

V is a normalised vector pointing at the viewer

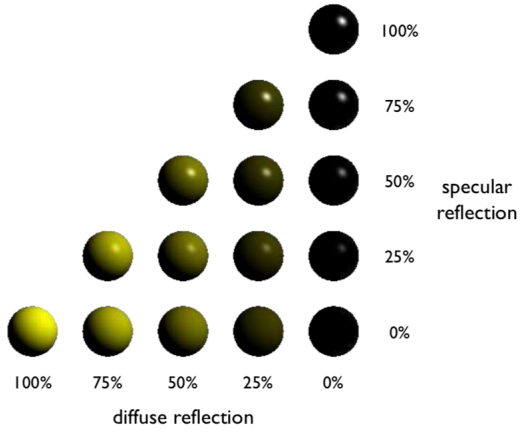
I_l is the intensity of the light source

k_s is the proportion of light which is specularly reflected by the surface

n is Phong's *ad hoc* "roughness" coefficient

I is the intensity of the specularly reflected light

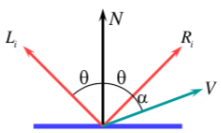
Examples



Shading: Overall reflection

The overall shading is the ambient illumination plus the diffuse and specular reflections from each light source.

$$I = I_a k_a + \sum_i I_i k_d (L_i \cdot N) + \sum_i I_i k_s (R_i \cdot V)^n$$



Sampling

So far, we have assumed that each ray passes through the centre of the pixel, therefore the value of the pixel is the colour of the object which lies under the centre of the pixel.

This leads to:

1. Stair step (jagged) edges to objects
2. Small objects being missed entirely
3. Thin objects being missed or split into small pieces.

Anti-aliasing

These artefacts (and similar ones) are known as aliasing – the methods of removing the effects of aliasing are known as anti-aliasing

In signal processing, aliasing is a precisely defined technical term for a particular kind of artefact, however in computer graphics, its meaning has expanded to include most undesirable effects that can occur in the image. **This is because some of the techniques which remove some of the effects of aliasing will remove others!**

Sampling in Ray Tracing

1. Single Point
    a. Shoots a single ray through the pixel's centre
2. Super-sampling (for anti-aliasing)
    a. Shoot multiple rays through the pixel and average the result
    b. The multiple rays are done through a number of different ways
        1. **Regular Grid**
            a. Divide the pixel into a number of sub-pixels and shoot a ray through each of their centres
            b. However, this can still lead to noticeable aliasing unless a very high resolution sub-pixel grid is used.
        2. **Random**
            a. Shoot pixels at random points in the pixel
            b. Replaces aliasing artefacts with noise artefacts **(however, the eye is far less sensitive to noise than to aliasing)**
        3. **Poisson Disc**
            a. Shoot random points with the proviso that no two rays through the pixel are closer than a set value from each other.
            b. This produces a better-looking image than pure random
            c. However, it is very hard to implement properly.
        4. **Jittered**
            a. Divide pixel into a number of sub-pixels and shoot one ray at a random point in each subpixel – this is an approximation to Poisson disc sampling
            b. **Better than random sampling and easy to implement.**
3. Adaptive super-sampling
    a. Shoot a few rays through the pixel, check the variance of the resulting values
    b. If they are similar (variance is low) – then stop, otherwise shoot some more rays.

Why take multiple samples per pixel: super-sampling is only one reason why we might take multiple sampler per pixel. You can get many effects by distributing the multiple samples over some range – **distributed ray tracing.** This works in one of a couple of ways:

1. Each of the multiple rays shot through a pixel is allocated a random value from the relevant distribution – all effects can be achieved this way with sufficient rays per pixel
2. Or each ray spawns multiple rays when it hits an object

Uses of Distributed Ray Tracing:

1. Distribute the samples for a pixel over the pixel area – used for antialiasing (as described)
2. Distribute the rays going to a light source over some area – allows area light sources in point and directional sources
    a. Produces softer shadows with penumbrae
3. Distribute the camera position over some area – allows the simulation of a camera with a finite aperture lens
4. Distribute the samples in time
    a. Produces motion blur effects on any moving objects

Graphics Pipeline

Since Ray Tracing is computationally so very expensive (only used by hobbyists and for super-high visual quality), we need a faster method.

Therefore, we can:

1. Model surfaces as polyhedra – meshes of polygons
2. Use composition to build scenes
3. Apply perspective transformation and project into the plane of the screen
4. Work out which surface is closest and fill the pixels with the colour of the nearest visible polygon

Since we normally do this, most modern graphics cards have hardware to support this.

Three-Dimensional Objects

Polyhedral surfaces are made up from meshes of multiple connected polygons. **Polygonal meshes are** open or closed and **are manifold or non-manifold.** A mesh is manifold if each edge is incident to only one or two faces (two points at the end of the edge is shared – no points along the edge).

In order to represent a curved surface, we must convert the entire surface into polygons which we can represent in memory. This means approximating the curve.

Triangles

We generally consider triangles since they must be planar, whereas any polygons which are larger than 4 are not necessarily planar and therefore the rotations and other transformations become both ambiguous and may not necessarily do what is predicted. Additionally, the computer hardware (GPUs) is geared towards working with triangles (for this reason) and therefore it is much faster to consider triangles.

Splitting polygons into triangles: This means we generally split all polygons with more than three vertices into triangles – however, it is to be noted that there are often a number of ways that we can split them – we should often consider which is better.

2D Transformations

- scale
  - about origin
  - by factor m

$$x' = mx$$
$$y' = my$$

- rotate
  - about origin
  - by angle θ

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

- translate
  - along vector (x_o, y_o)

$$x' = x + x_o$$
$$y' = y + y_o$$

- shear
  - parallel to x axis
  - by factor a

$$x' = x + ay$$
$$y' = y$$

- scale
  - about origin, factor m

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} m & 0 \\ 0 & m \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- rotate
  - about origin, angle θ

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- do nothing
  - identity

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- shear
  - parallel to x axis, factor a

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

However, translations cannot be represented using a simple 2D matrix multiplication, so we switch to **homogeneous 2D co-ordinates**, where an infinite number of homogeneous co-ordinates map to every 2D point – and w=0 represents a point at infinity.

$$(x, y, w) \equiv \left(\frac{x}{w}, \frac{y}{w}\right)$$

We usually take the inverse transform (normal to homogeneous) to be (x, y) = (x, y, 1)

Matrices in homogeneous co-ordinates

- **scale**
  - **about origin, factor** m

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- **rotate**
  - **about origin, angle** θ

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- **do nothing**
  - **identity**

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- **shear**
  - **parallel to** x **axis, factor** a

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Translation

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_o \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

In homogeneous coordinates

$$x' = x + wx_o \qquad y' = y + wy_o \qquad w' = w$$

Concatenating Transformations

We can concatenate transformations by multiplying their matrices (in reverse order to the order in which they are to be performed).

This lets us do things like scale / rotate about an arbitrary point by:

1. Translating the point to the origin
2. Scale / rotate
3. Translate back

3D Transformations

Similarly to 2D transformations:

3D homogeneous co-ordinates

$$(x, y, z, w) \rightarrow \left(\tfrac{x}{w}, \tfrac{y}{w}, \tfrac{z}{w}\right)$$

3D transformation matrices

translation

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

identity

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

rotation about *x*-axis

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

scale

$$\begin{bmatrix} m_x & 0 & 0 & 0 \\ 0 & m_y & 0 & 0 \\ 0 & 0 & m_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotation about z-axis

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

rotation about *y*-axis

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

Model Transformation 1:

We always scale first, then rotate and then translate.

Rotation: Rotation is a multi-step process. We break rotation into steps each of which is a rotation about a principal axis. We work these out by taking the desired orientation back to the original axis-aligned position.

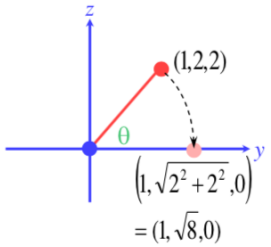
EXAMPLE:

- desired axis: (2,4,5)–(1,2,3) = (1,2,2)
- original axis: y-axis = (0,1,0)

- zero the z-coordinate by rotating about the x-axis

$$\mathbf{R}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

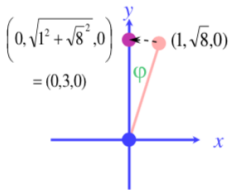
$$\theta = -\arcsin\frac{2}{\sqrt{2^2+2^2}}$$



- then zero the x-coordinate by rotating about the z-axis
- we now have the object's axis pointing along the y-axis

$$\mathbf{R}_2 = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\varphi = \arcsin\frac{1}{\sqrt{1^2+\sqrt{8}^2}}$$



- the overall transformation is:
  - first scale
  - then take the inverse of the rotation we just calculated
  - finally translate to the correct position

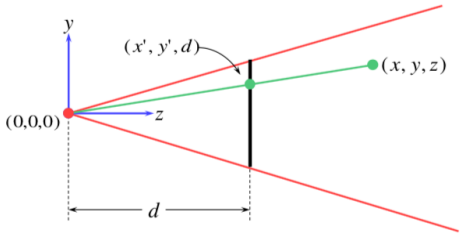
$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \mathbf{T} \times \mathbf{R}_1^{-1} \times \mathbf{R}_2^{-1} \times \mathbf{S} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

3D => 2D Projection

In order to make a picture, we project a 3D world to a 2D image, by projecting the three-dimensional world being projected onto a plane.

There are a couple of types of projection, using:

1. Parallel
   a. (x, y, z) -> (x, y)
   b. This is used in CAD, architecture, etc (lengths are preserved)
   c. However, it looks unrealistic.
2. Perspective
   a. (x, y, z) -> (x/z, y/z)
   b. Things get smaller as they get further away
   c. Things look realistic – this is how cameras work.



$$x' = x\frac{d}{z}$$

$$y' = y\frac{d}{z}$$

Projection as a matrix operation

$$\begin{bmatrix} x \\ y \\ 1/d \\ z/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$x' = x\frac{d}{z}$$

$$y' = y\frac{d}{z}$$

remember $\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \begin{bmatrix} x/w \\ y/w \\ z/w \end{bmatrix}$

This is useful in the z-buffer algorithm where we need to interpolate $1/z$ values rather than z values.

$$z' = \frac{1}{z}$$

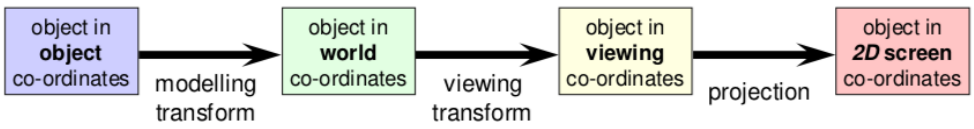
Perspective Projection with an Arbitrary Camera

Assumptions are:

1. Screen centre is at (0, 0, d)
2. The screen is parallel to the xy-plane
3. The z-axis is into the screen
4. The y-axis is up and the x-axis to the right
5. The camera is at the origin

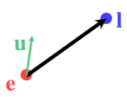
We can either work out a equations for projecting objects about an arbitrary point onto an arbitrary point **or transform all objects into our standard coordinate system (viewing coordinates) and use the above assumptions.**

MVP Transformations



Viewing Transformation

- the problem:
  - to transform an arbitrary co-ordinate system to the default viewing co-ordinate system
- camera specification in world co-ordinates
  - eye (camera) at (e_x,e_y,e_z)
  - look point (centre of screen) at (l_x,l_y,l_z)
  - up along vector (u_x,u_y,u_z)
    - perpendicular to el



- First, we translate the eye-point to the origin and scale so that eye point to look point vector is the distance from the origin to screen centre d.
- Then we rotate so the line el is aligned with the z-axis.

◆ translate eye point, (e_x,e_y,e_z), to origin, $(0,0,0)$

$$\mathbf{T}=\begin{bmatrix}1&0&0&-e_x\\0&1&0&-e_y\\0&0&1&-e_z\\0&0&0&1\end{bmatrix}$$

◆ scale so that eye point to look point distance, $|\overline{\mathbf{el}}|$, is distance from origin to screen centre, d

$$|\overline{\mathbf{el}}|=\sqrt{(l_x-e_x)^2+(l_y-e_y)^2+(l_z-e_z)^2} \qquad \mathbf{S}=\begin{bmatrix}d/|\overline{\mathbf{el}}|&0&0&0\\0&d/|\overline{\mathbf{el}}|&0&0\\0&0&d/|\overline{\mathbf{el}}|&0\\0&0&0&1\end{bmatrix}$$

◆ need to align line $\overline{\mathbf{el}}$ with z-axis

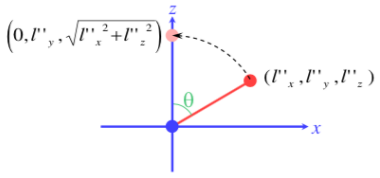
- first transform e and l into new co-ordinate system

$$\mathbf{e}''=\mathbf{S}\times\mathbf{T}\times\mathbf{e}=\mathbf{0} \qquad \mathbf{l}''=\mathbf{S}\times\mathbf{T}\times\mathbf{l}$$

- then rotate e''l'' into yz-plane, rotating about y-axis

$$\mathbf{R}_1=\begin{bmatrix}\cos\theta&0&-\sin\theta&0\\0&1&0&0\\\sin\theta&0&\cos\theta&0\\0&0&0&1\end{bmatrix}$$

$$\theta=\arccos\frac{l''_z}{\sqrt{l''_x{}^2+l''_z{}^2}}$$

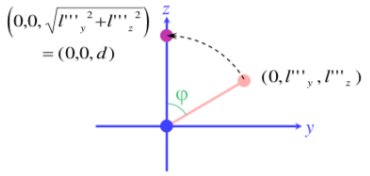


◆ having rotated the viewing vector onto the yz plane, rotate it about the x-axis so that it aligns with the z-axis

$$\mathbf{l}'''=\mathbf{R}_1\times\mathbf{l}''$$

$$\mathbf{R}_2=\begin{bmatrix}1&0&0&0\\0&\cos\varphi&-\sin\varphi&0\\0&\sin\varphi&\cos\varphi&0\\0&0&0&1\end{bmatrix}$$

$$\varphi=\arccos\frac{l'''_z}{\sqrt{l'''_y{}^2+l'''_z{}^2}}$$



◆ the final step is to ensure that the up vector actually points up, i.e. along the positive y-axis

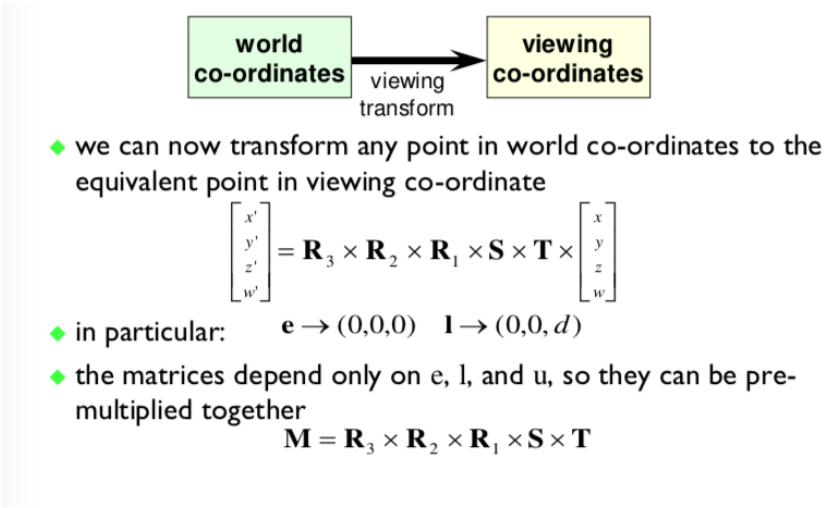
- actually need to rotate the up vector about the z-axis so that it lies in the positive y half of the yz plane

$$\mathbf{u}''''=\mathbf{R}_2\times\mathbf{R}_1\times\mathbf{u}$$

$$\mathbf{R}_3=\begin{bmatrix}\cos\psi&-\sin\psi&0&0\\\sin\psi&\cos\psi&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$\psi=\arccos\frac{u''''_y}{\sqrt{u''''_x{}^2+u''''_y{}^2}}$$

Therefore,



Illumination and Shading

If we draw polygons with uniform colours around them, we generally get very poor results therefore we must interpolate the colours across the polygons.

In order to interpolate the colours across the polygons, we need:

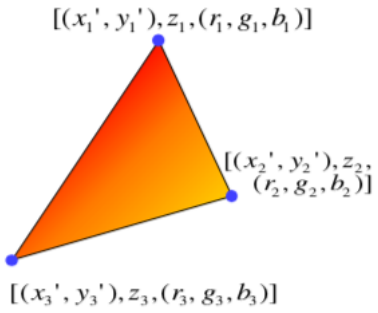
1. A colour at each vertex
2. An algorithm to blend between the colours across the vertex
3. **Normals at each point**
    a. Specular reflection requires this – ambient and diffuse can just work off the first two.

Gouraud Shading

We calculate the diffuse illumination at each vertex by:

- Calculating normal at the vertex and use this to calculate the diffuse illumination at this point
- Normal can be calculated directly.

Then we interpolate the colour between the vertices of the polygon. This means the surface will look smoothly curved, rather than looking like a set of polygons – though the outline will still look polygonal.

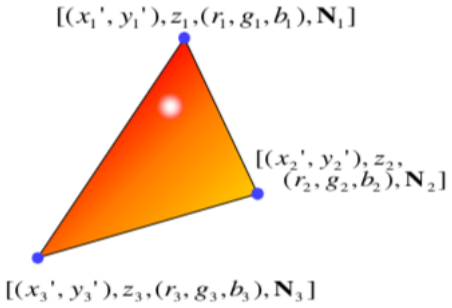


While much better than flat shading it still has the flaw of missing specular highlights – therefore we must think even more carefully.

Phong Shading
The Phong shading is similar to Gouraud shading but calculates the specular components in addition to the diffuse component.

Therefore, we interpolate the normal across the polygon in order to calculate the reflection vector. Then for each point, we calculate the specular reflection and add in the diffuse illumination.

N.B. Phong's approximation to specular reflection ignores (amongst other things), the effects of glancing incidence.



Graphics Hardware and OpenGL

Graphics Processing Unit (GPU)

A GPU is like a CPU (Central Processing Unit) for processing graphics. It is optimised for floating point operations on large arrays of data – with vertices, normal, pixels, etc.

It performs all low-level tasks and a lot of high-level tasks:

- Clipping, rasterization, hidden surface removal – essentially draws triangles very efficiently
- Procedural Shading, texturing, animation, simulation etc
- Video rendering, de and encoding, de-interfacing
- Physics engines

It generally offers full programmability at several pipeline stages – but it is optimised for massively parallel operations.

Why is a GPU fast?

- 3D rendering can be efficiently parallelized
    - Allow us to complete millions of pixels, millions of triangles, all of which have the operations which can be computed independently at the same time.
- Hence, modern GPUs contain lots of SIMD (Single Instruction Multiple Data) which operates on large arrays of data
- It has >> 400 GB/s which has a much higher bandwidth than CPU – but the peak performance can be expected for specific operations

GPU APIs

OpenGL

- Multiplatform
- Open standard API
- Has a focus on general 3D applications – the OpenGL driver manages the resources
- **OpenGL ES**
    - OpenGL ES is a stripped version of OpenGL – removes the functionality that is not necessary on mobile devices
    - It is made for a whole range of devices including:
        - iOS
        - Android
        - Playstation 3
        - Nintendo 3DS
- **OpenGL in Java**
    - The Standard Java API does not include OpenGL interface
    - But several wrapper libraries exist including
        - Java OpenGL (JOGL)
        - Lightweight Java Game Library – LWJGL
            - We use this (version 3)
            - It is better maintained
            - It has access to other APIs (OpenCL, OpenAL, …)
    - We also need to use a linear algebra library
        - JOML (Java OpenGL Math Library)
        - It operates on 2, 3, 4-dimensional vectors and matrices
- **OpenGL History**
    - It originated as a proprietary library IRIS GL made by SGI
    - OpenGL 1.0 (1992)
    - OpenGL 1.2 (1998)
    - OpenGL 2.0 (2004)
        - GLSL
        - Non-power-of-two textures (NPOT)
    - OpenGL 3.0 (2008)
        - It was a major overhaul of the API
        - Many features from previous versions were deprecated.
    - OpenGL 3.2 (2009)
        - Core and Compatability profiles
        - Geometry Shaders
    - OpenGL 4.0 (2010)
        - Catching up with Direct3D 11
    - OpenGL 4.5 (2014)
    - OpenGL 4.6 (2017)

DirectX

- Microsoft Windows / Xbox
- Proprietary API
- It has a focus on games – the application manages the resources.

Vulkan

- It is cross platform and an open standard

- It is a very low-overhead API for high performance 3D graphics compared to OpenGL and DirectX
    - It reduces the CPU load and works better for multi-CPU-core architectures
    - It also gives much finer control of the GPU
- But
    - The code for drawing a few primitives can take 1000s of lines of code
    - It is intended for game engines and code that must be very well optimised.

CUDA

- OpenGL and DirectX are not meant to be used for general purpose computing – physical simulation, machine learning
- CUDA is NVidia's architecture for parallel computing
    - C-like programming language
    - With special API for parallel instructions
    - Requires NVidia GPU

OpenCL

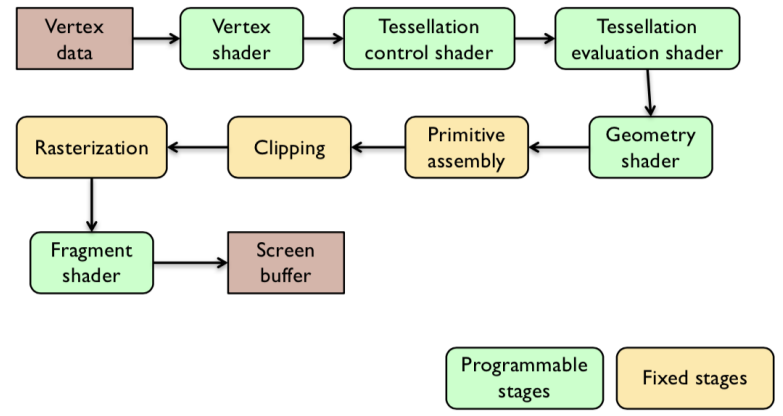
- Similar to CUDA but it is an open standard
- It can run on both the GPU and the CPU
- It is supported by AMD, Intel and NVidia, Qualcomm, Apple …
- Very widely used therefore,

OpenGL Rendering Pipeline

Model

1. CPU Code
    - gl* functions that
        - Create OpenGL objects
        - Copy data between the CPU and the GPU
        - Modify the OpenGL state
        - Enqueue operations
        - Synchronise CPU and GPU
    - C99 Library
    - Wrappers in most programming languages

2. GPU Code
    - Fragment Shaders
    - Vertex Shaders
    - Other shaders
    - It is written in GLSL
        - Similar to C
        - From OpenGL 4.6, they could be written in other languages and compiler to SPIR-V



Vertex Shader
Processes vertices, normal and (u, v) texture coordinates.

Tessellation Control Shader & Tessellation Evaluation Shader
Create new primitives by tessellating existing primitives (patches).

Geometry Shader
This is an optional step – it operates on tessellated geometry. It can create new primitives.

Primitive Assembly
It organises vertices into primitives and prepares them for rendering.

Clipping
It removes vertices so they all lie within the viewport (view frustrum). This means that new primitives might need to be created.

Rasterization
It generates fragments (pixels) to be drawn for each primitive.
It interpolates vertex attributes.

Fragment Shader
Computes the colour for each fragment (pixel). It can look up the colour in the texture and can modify the pixels' depth value.

Working with Buffers

Generating Names

- "name" is like a reference in Java
- glGen* functions create names WITHOUT allocating the actual object.
- From OpenGL 4.5 glCreate* functions create names and allocate the actual object.

Binding Objects

- glBind* functions
- Performs two operations
    - Allocates memory for the particular object (if it does not exist)
    - Makes it active in the current OpenGL context

- Functions operating on OpenGL objects will change the currently bound (or active) object.

Unbinding Objects

- Passing "0" instead of "name" unbinds the active object.
    - glBind(..., 0)

Deleting Objects

- glDelete* functions delete both the objects and its name.

When dealing with buffers, you must be aware that OpenGL is not an Object-Oriented API so we must do things like:

```
int va = glGenVertexArrays();
glBindVertexArray(va); // va
becomes "active" VertexArray

int vertices = glGenBuffers();
glBindBuffer(GL_ARRAY_BUFFE
R, vertices); // This adds vertices
to currently bound VertexArray
```

Example OpenGL Code

In order to draw some triangles, we must:

1. Initialise OpenGL
    a. Initialising rendering windows and OpenGL context
    b. Send the geometry (vertices, triangles, normals) to the GPU
    c. Load and compile Shaders
2. Set up inputs
3. Draw a frame
    a. Clear the screen buffer
    b. Set the MVP (model view projection) matrix
    c. Render geometry
    d. Flip the screen buffers
4. Free resources

Initialise

```
int vertexArrayObj = glGenVertexArrays(); // Create a name
glBindVertexArray(vertexArrayObj);          // Bind a VertexArray

float[] vertPositions = new float[] { -1, -1, 0,  0, 1, 0,  1, -1, 0 }; // x, y, z, x, y, z …
// Java specific code for transforming float[] into an OpenGL-friendly format
FloatBuffer vertex_buffer = BufferUtils.createFloatBuffer(vertPositions.length);
vertex_buffer.put(vertPositions); // Put the vertex array into the CPU buffer
vertex_buffer.flip();                  // "flip" is used to change the buffer from read to
write mode

int vertex_handle = glGenBuffers();    // Get an OGL name for a buffer object
glBindBuffer(GL_ARRAY_BUFFER, vertex_handle); // Bring that buffer object into
existence on GPU
glBufferData(GL_ARRAY_BUFFER, vertex_buffer, GL_STATIC_DRAW); // Load the
GPU buffer object with data
```

Rendering

```
// Step 1: Pass a new model-view-projection matrix to the vertex shader
Matrix4f mvp_matrix; // Model-view-projection matrix
mvp_matrix = new
Matrix4f(camera.getProjectionMatrix()).mul(camera.getViewMatrix());

int mvp_location = glGetUniformLocation(shaders.getHandle(), "mvp_matrix");
FloatBuffer mvp_buffer = BufferUtils.createFloatBuffer(16);
mvp_matrix.get(mvp_buffer);
glUniformMatrix4fv(mvp_location, false, mvp_buffer);

// Step 2: Clear the buffer
glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // Set the background colour to dark grey
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
// Step 3: Draw our VertexArray as triangles
glBindVertexArray(vertexArrayObj); // Bind the existing VertexArray object
glDrawElements(GL_TRIANGLES, no_of_triangles, GL_UNSIGNED_INT, 0); //
Draw it as triangles
glBindVertexArray(0); // Remove the binding

// Step 4: Swap the draw and back buffers to display the rendered image
glfwSwapBuffers(window);
glfwPollEvents();
```

GLSL Fundamentals

Shaders

They are small programs executed on a GPU. They are executed for every vertex, each fragment etc.

Data Types

- Basic types
  - float, double, int, uint, bool
- Aggregate types
  - float: vec2, vec3, vec4; mat2, mat3, mat4
  - double: dvec2, dvec3, dvec4; dmat2, dmat3, dmat4
  - int: ivec2, ivec3, ivec4
  - uint: uvec2, uvec3, uvec4
  - bool: bvec2, bvec3, bvec4

```
vec3 V = vec3( 1.0, 2.0, 3.0 );   mat3 M = mat3( 1.0, 2.0, 3.0,
                                                 4.0, 5.0, 6.0,
                                                 7.0, 8.0, 9.0 );
```

Indexing Components

You can index components of aggregate types very easily, in a number of different ways.

- Subscripts: rgba, xyzw, stpq (work exactly the same)
  - float red = color.r;
  - float v_y = velocity.y;

  but also
  - float red = color.x;
  - float v_y = velocity.g;
- With 0-base index:
  - float red = color[0];
  - float m22 = M[1][1]; // second row and column of matrix M

Swizzling

You can also select the elements of the aggregate type:

- vec4 rgba_color(1.0, 1.0, 0.0, 1.0);
- vec3 rgb_color = rgba_color.rgb;
- vec3 bgr_color = rgba_color.bgr;
- vec3 luma = rgba_color.ggg;

Arrays

Arrays work exactly the same as for C:

```
float lut[5] = float[5]( 1.0, 1.42, 1.73, 2.0, 2.23 );
```

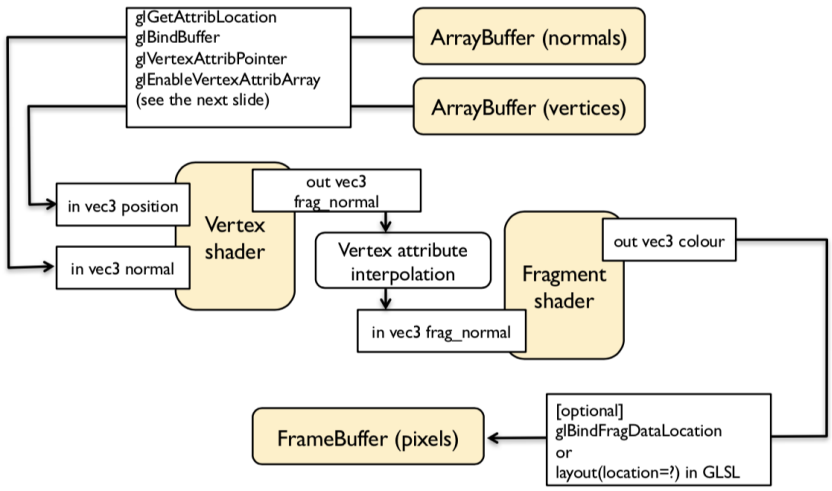
- Size can be checked with "length()"

```
for( int i = 0; i < lut.length(); i++ ) {
  lut[i] *= 2;
}
```

Storage Qualifiers

- `const` – read-only, fixed at compile time
- `in` – input to the shader
- `out` – output from the shader
- `uniform` – parameter passed from the application (Java), constant for the primitive
- `buffer` – shared with the application
- `shared` – shared with local work group (compute shaders only)

Shader Inputs and Outputs



We specify input to a vertex shader by calling a 'function' with the parameters of the variable that the shader should have.

For inputs, we add it to a buffer, enabling it and telling it where and of what form the variables are before enabling the variable:

```
// Get the locations of the "position" vertex attribute variable
in our shader
int position_loc = glGetAttribLocation(shaders_handle,
"position");
// If the vertex attribute found
if (position_loc != -1) {
  // Activate the ArrayBuffer that should be accessed in the
shader
  glBindBuffer(GL_ARRAY_BUFFER, vertex_handle);
  // Specifies where the data for "position" variable can be
accessed
  glVertexAttribPointer(position_loc, 3, GL_FLOAT, false, 0, 0);
  // Enable that vertex attribute variable
  glEnableVertexAttribArray(position_loc);
}
```

For uniforms, we simply define the uniform in the shader, before adding it to the buffer and giving the buffer to glsl.

- In shader:

```
uniform mat4 mvp_matrix; // model-view-projection matrix
```

- In Java:

```
Matrix4f mvp_matrix; // Matrix to be passed to the shader
...
int mvp_location = glGetUniformLocation(shaders.getHandle(),
                   "mvp_matrix");
FloatBuffer mvp_buffer = BufferUtils.createFloatBuffer(16);
mvp_matrix.get(mvp_buffer);
glUniformMatrix4fv(mvp_location, false, mvp_buffer);
```

Name of the method depends on the data type.
For example, glUniform3fv for Vector3f

Operators

- Arithmetic: + - ++ --
  - Multiplication:
    - vec3 * vec3 – element-wise
    - mat4 * vec4 – matrix multiplication (with a column vector)
- Bitwise (integer): <<, >>, &, |, ^
- Logical (bool): &&, ||, ^^
- Assignment:

```
float a=0;
a += 2.0; // Equivalent to a = a + 2.0
```

Math

- Trigonometric:
  - `radians( deg ), degrees( rad ), sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, asinh, acosh, atanh`
- Exponential:
  - `pow, exp, log, exp2, log2, sqrt, inversesqrt`
- Common functions:
  - `abs, round, floor, ceil, min, max, clamp, …`
- And many more

Flow Control

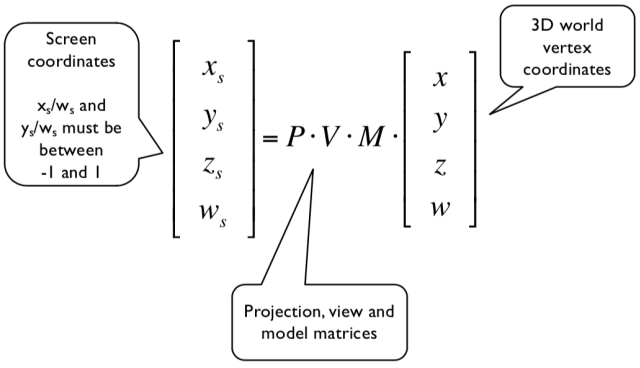
```
if( bool ) {
  // true
} else {
  // false
}

switch( int_value ) {
  case n:
    // statements
    break;
  case m:
    // statements
    break;
  default:
}

for( int i = 0; i<10; i++ ) {
   ...
}

while( n < 10 ) {
 ...
}

do {
 ...
} while ( n < 10 )
```

Transformations and Vertex Shaders

In order to go from **Objects coordinates to World Coordinates**, you need to use the Model matrix to change the object. It could be different for every object.

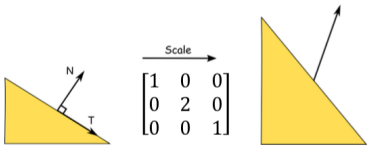
In order to go from **World coordinates to View coordinates (where the camera is at the origin, pointing at –z) you need the view matrix.** This is common for all objects in the scene.

Finally, to go from the **View coordinates to the Screen coordinates (where x and y are in range -1 to 1) and the entire scene is projected to a '2D plane') you use the projection matrix which is defined for every object.** It is important to note that the z coordinate is still maintained in this step since it is required for depth testing (which object is in front).



Transforming Normal Vectors

- Transformation by a nonorthogonal matrix does not preserve angles



- Since: $N \cdot T = 0$

Normal transform

$$N' \cdot T' = (GN) \cdot (MT) = 0$$

Transformed normal and tangent vector

Vertex position transform

- We can find that: $G = (M^{-1})^T$
  - Derivation shown on the visualizer

Raster Buffers

Render Buffers

Colour:	GL_FRONT	GL_BACK	Four components: RGBA Typically 8 bits per component
In stereo:	GL_FRONT_LEFT	GL_FRONT_RIGHT	
	GL_BACK_LEFT	GL_BACK_RIGHT	
Depth:	DEPTH		To resolve occlusions (see Z-buffer algorithm) Single component, usually >8 bits
Stencil:	STENCIL		To block rendering selected pixels Single component, usually 8 bits.

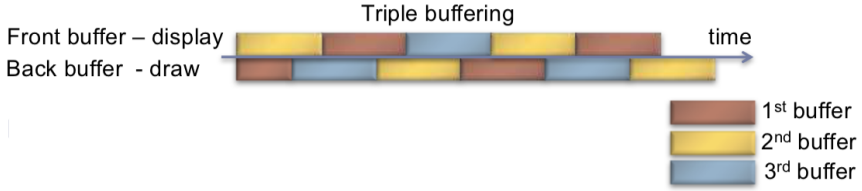
Double Buffering

In order to avoid flicker and tearing, two buffers (rasters) are used:

- The front buffer shows what is shown on the screen
- The back buffer is not shown and the GPU draws into that buffer

Then, when the drawing is finished, you swap the front and back buffers – ie then use the 2nd buffer to display and use the first to draw in.

However, by introducing a **third buffer**, you can eliminate the gaps which reduce the efficiency of the system.



However, while more efficient this has two issues:

1. Requires more memory
2. Leads to a larger delay between drawing a displaying a frame – though this could happen quicker per frame.

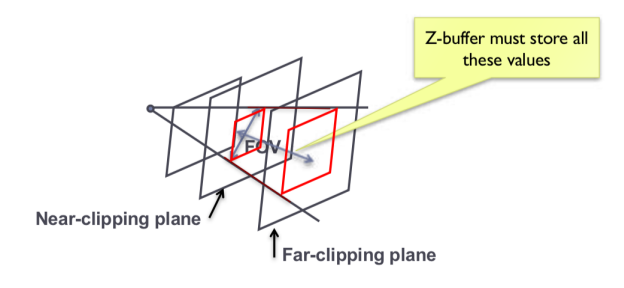
Z-Buffer

1. Initialise the depth buffer and image buffer for all pixels
    a. Color (x, y) = BackgroundColour
    b. Depth (x, y) = z_far // position of far clipping area.
2. For every triangle
    a. Calculate z for current (x, y)
    b. If (z < Depth (x, y))
        i. Depth (x, y) = z
        ii. Color (x, y) = Polygon_Color (x, y)

Clipping

You define the view frustrum by defining the near-clipping plane and the far-clipping plane which defines the objects which are visible.

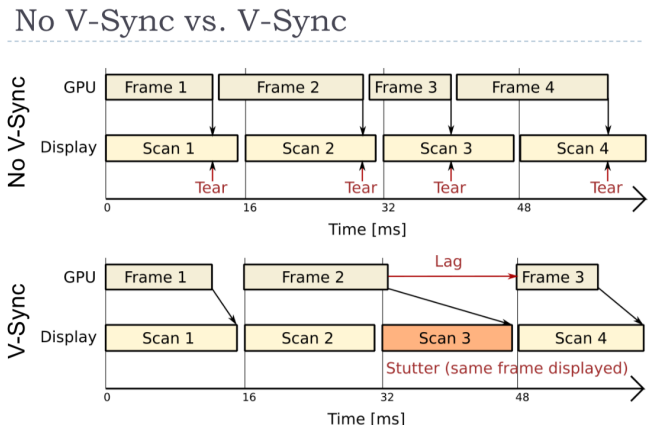
It is a frustrum since you are observing from a set point (camera).



Vertical Synchronisation: V-Sync

Since pixels are copied from the colour buffer to the monitor row-by-row, you can get **tearing artefacts.** This occurs when the front and back buffer are swapped in the middle of the process of copying – therefore the upper part of the screen contains the previous frame and the lower part contains the current frame.

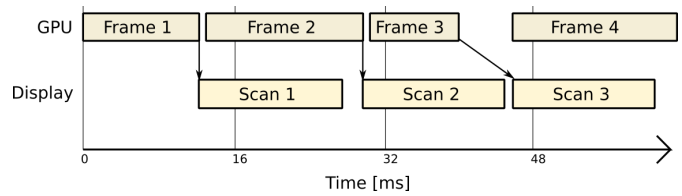
In OpenGL, you can use the glwfSwapInterval(1) command to wait until the last raw is copied to the display before swapping the buffers.



Note, here, the scan happens a set number of times per second.

FreeSync (AMD) and G-Sync (Nvidia)

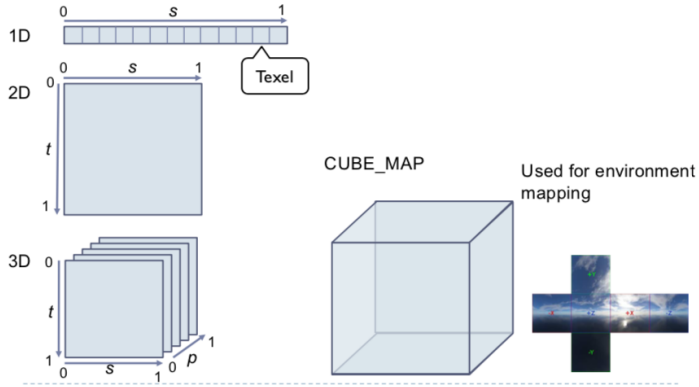
This makes use of adaptive sync, where the graphics card controls the timing of frames on the display. This can save power where useful and also reduce lag for real-time graphics.



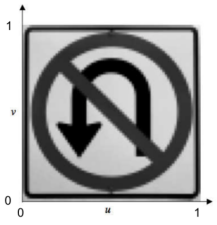
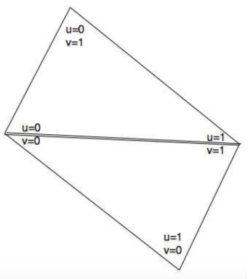
Textures

OpenGL Texture Types

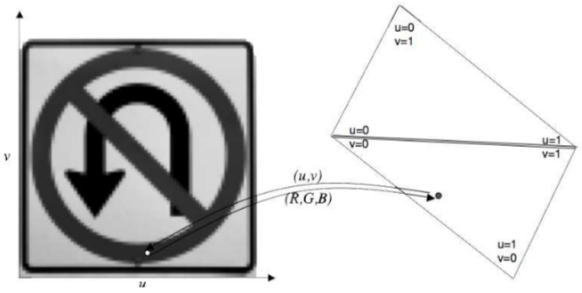
1. In **1D**, it is effectively an array of texels
    a. Therefore, textures can have any size but the sizes that are powers of 2 (2^n) may give better performance.
2. In **2D,** it is an array of arrays of texels.
3. In **3D,** it is an array of 2D textures.
4. In a **Cube Map,** we represent a texture of a surface of a cube using the map of it.



Texture Mapping

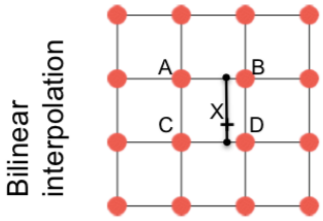
1. Define a texture mapping function (image) – T(u, v) where (u, v) is the texture coordinates.
    a. 
2. Define the correspondence between the vertices on the 3D object and the texture coordinates.
    a. 

3. For every surface point compute texture coordinates – using the texture function to get the texture value. This gets used as a colour or reflectance.

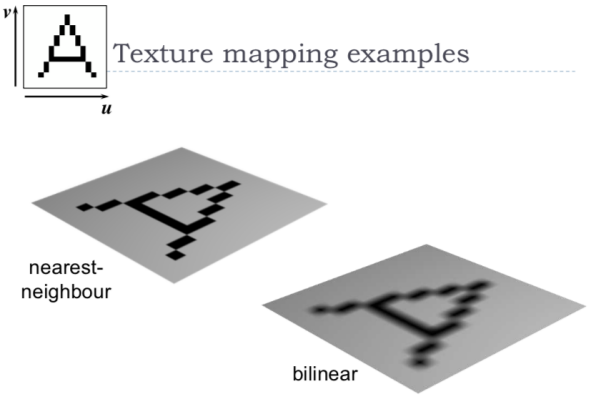
a. 

Sampling

Nearest neighbour sampling: Pick the texture of the nearest texel.
Bilinear Interpolation: Interpolate to find the texture of the point – first interpolating for the x-axis and then interpolating for the y-axis.



Interpolate first along x-axis between AB and CD, then along y-axis between the interpolated points.

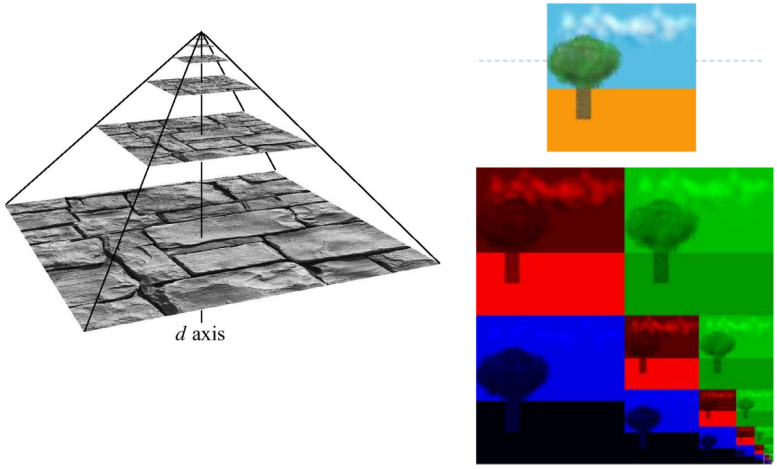


1. **Up-sampling**
    a. More pixels than texels
    b. Values need to be interpolated.
    c. **You will always get visible artefacts – the only way to prevent this is to ensure that the texture map is of a sufficiently high resolution so it doesn't happen.**
    d. **Nearest-Neighbour**
        i. You get blocky artefacts
    e. **Bilinear**

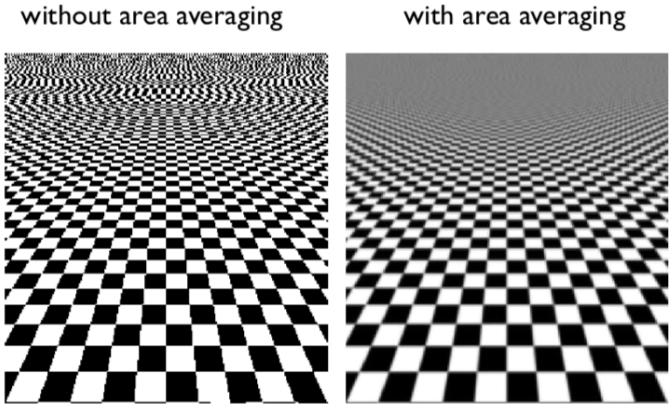
   i. You get blurry artefacts
2. **Down-sampling**
   a. There are fewer pixels than texels
   b. The values need to be averaged over an area of the texture
      i. Using a **mipmap**
      ii. You need to average the texture across that area, not just take a sample in the middle of the area.

Mipmap

Textures can be stored at multiple resolutions as a mipmap – each level of the pyramid is half the size of the lower level. It provides pre-filtered texture (area-averaged) when screen pixels are larger than the full resolution texels. The mipmap requires a third of the original texture size to store.



In order to use the mipmap, we find the level that it has a 1-1 mapping and combine the three items and recombine them to form a single texture map.



Texture Tiling: Repetitive patterns can be represented as texture tiles. It is made so that the texture folds over.

Texture Atlas: A single texture can be used and referenced for multiple surfaces and objects.

Bump Mapping

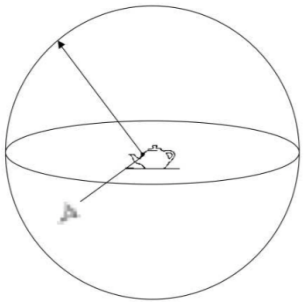
Bump mapping is a special kind of texture that modifies the surface normal (a vector that is perpendicular to a surface). The surface is still flat but shading appears as on an uneven surface. This is easily done **in fragment shaders.**

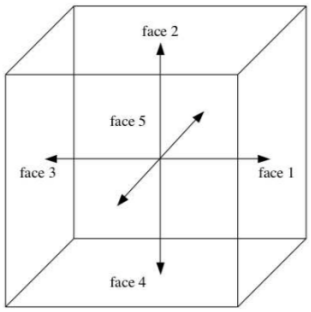
Displacement Mapping

Displacement Mapping is a texture that modifies the surface. It has better results than bump mapping since the surface is not flat. This requires **geometry shaders.**

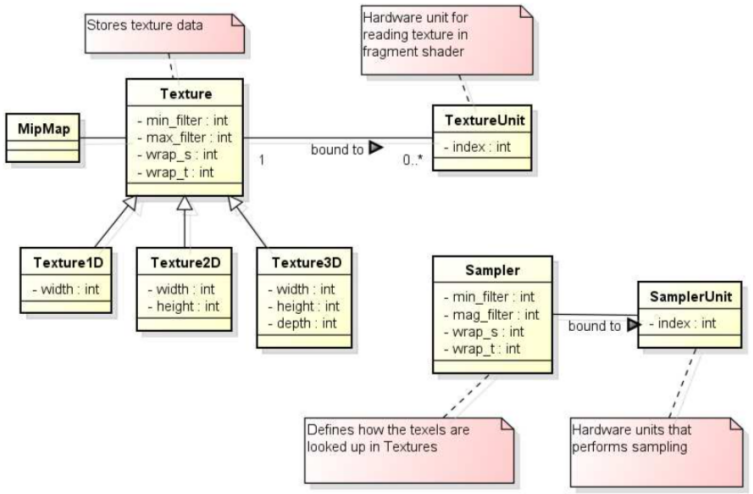
Environment Mapping

We use environment mapping to show the environment reflected by an object. In order to do this, we use an environment cube – with each face capturing the environment in that direction and therefore we can recombine that to show something.





Texture Objects in OpenGL



Setting up a texture

```
// Create a new texture object in memory and bind it
int texId = glGenTextures();
glActiveTexture(textureUnit);
glBindTexture(GL_TEXTURE_2D, texId);

// All RGB bytes are aligned to each other and each component is
1 byte
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

// Upload the texture data and generate mipmaps
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, tWidth, tHeight, 0,
        GL_RGBA, GL_UNSIGNED_BYTE, buf);
glGenerateMipmap(GL_TEXTURE_2D);
```

Texture parameters

```
//Setup filtering, i.e. how OpenGL will interpolate the pixels
when scaling up or down
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_NEAREST);
```

How to interpolate in 2D

How to interpolate between mipmap levels

```
//Setup wrap mode, i.e. how OpenGL will handle pixels outside of
the expected range
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);
```

Fragment shader

```
#version 330
uniform sampler2D texture_diffuse;
in vec2 frag_TextureCoord;

out vec4 out_Color;

void main(void) {
        out_Color = texture(texture_diffuse, frag_TextureCoord);
}
```

Rendering

```
// Bind the texture
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texId);

glBindVertexArray(vao);
glDrawElements(GL_TRIANGLES, indicesCount, GL_UNSIGNED_INT, 0);
glBindVertexArray(0);

glBindTexture(GL_TEXTURE_2D, 0);
```

Frame Buffer Objects

Instead of rendering to the screen buffer (back buffer – GL_BACK), an image can be rendered to an off-screen buffer: this is a Texture or RenderBuffer. It is faster to render to a RenderBuffer than a Texture but it cannot be copied, pixels can only be copied.

They can be used for:

1. Post-processing, tone-mapping, blooming, etc.
2. Reflections (in water), animated textures.
3. When the result of rendering is not shown on the screen – it is saved to disk.

Using a FBO

1. Create a FBO object
2. Attach a Texture (colour) and a RenderBuffer (depth)

```
int color_tex = glGenTextures();
glBindTexture(GL_TEXTURE_2D, color_tex);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, 256, 256, 0, GL_BGRA,
GL_UNSIGNED_BYTE, NULL);

int myFBO = glGenFramebuffers();
glBindFramebuffer(GL_FRAMEBUFFER, myFBO);
//Attach 2D texture to this FBO
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
GL_TEXTURE_2D, color_tex, 0);
int depth_rb = glGenRenderbuffers();
glBindRenderbuffer(GL_RENDERBUFFER, depth_rb);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT24,
256, 256);
//Attach depth buffer to FBO
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
GL_RENDERBUFFER, depth_rb);
```

- Render

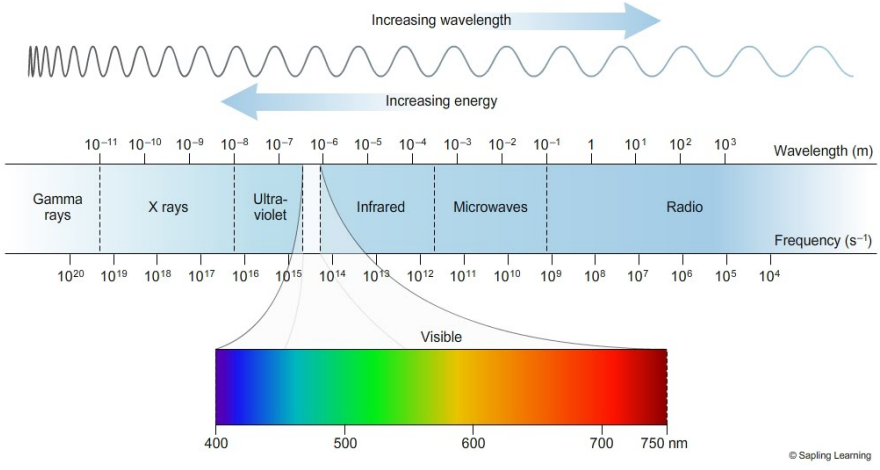
```
glBindFramebuffer(GL_FRAMEBUFFER, myFBO);
glClearColor(0.0, 0.0, 0.0, 0.0);
glClearDepth(1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// Render

glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Colour

Definitions

Electromagnetic Spectrum

Visible Light is EM waves of wavelength in the range of 380nm to 730nm. The earth's atmosphere lets a lot of light in this wavelength band through. It is higher in energy than thermal infrared so heat does not interfere with vision.



Colour

Colour is the result of our perception – there is no physical definition of colour.
However, for emissive displays / objects:

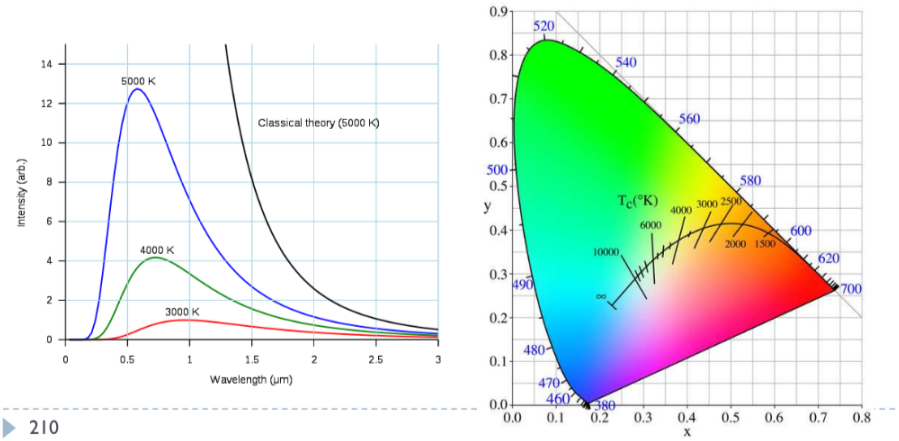
$$colour = perception\ (spectral\ emission)$$

For reflective displays / objects:

$$colour = perception\ (illumination * reflectance)$$

Black body radiation

Black body radiation is radiation emitted by a perfect absorber at a given temperature – Graphite is a good approximation of a black body.



210

It is related by:

$$W = \lambda_{max} T$$

where lambda$_{max}$ is the wavelength of maximum intensity.

Correlated colour temperature

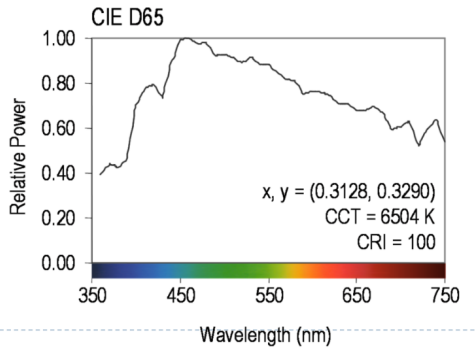
The correlated colour temperature is the temperature of a black body radiator that produces light most closely matching the particular source:

- Typical north-sky light – 7500K
- Typical average daylight – 6500K
- Domestic tungsten lamp (100 – 200W) – 2800K
- Domestic tungsten lamp (40 – 60W) – 2700K
- Sunlight at sunset – 2000K

It is used to describe the colour of the illumination – **illumination is the source of light.**

Standard illuminant D65

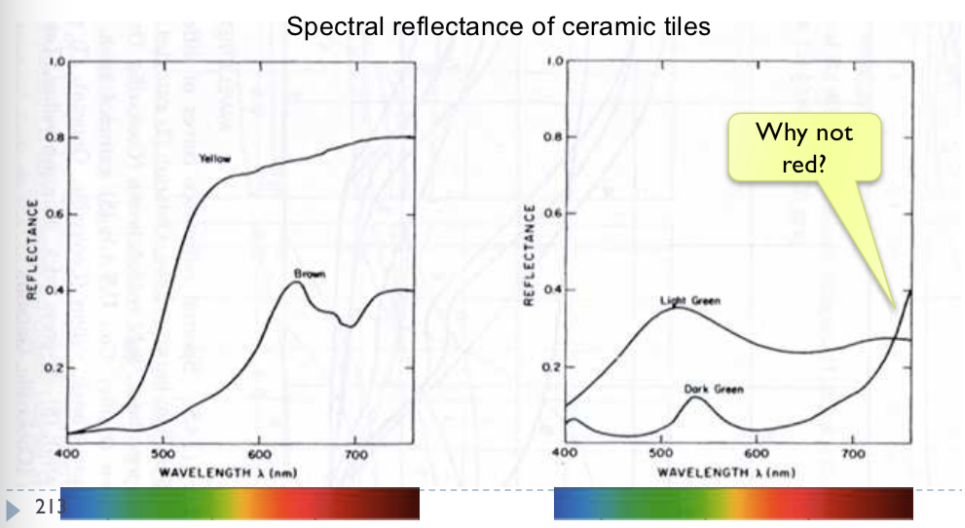
This is a standardises illuminant of midday sun in Western Europe / Northern Europe. This has a colour temperature of approximately 6500K.



Reflectance

Most of the light we see is reflected from objects – these objects absorb and reflect a certain part of the light spectrum.

Example:



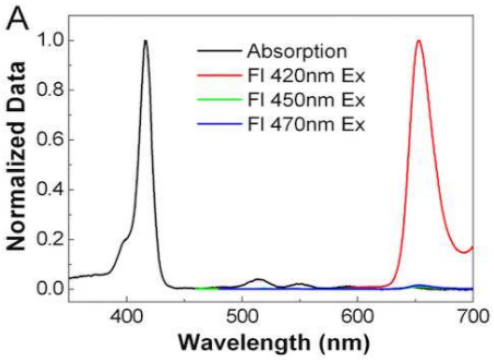
$$Reflected\ Light\ (L) = Illumination\ (I) * Reflectance\ (R)$$

This shows that the same object may appear to have different colours depending on the different illumination.

Fluorescence

Fluorescence is the emission of light by a substance that has absorbed light or other EM radiation. It is a form of **luminescence.** In most cases, the emitted light has a longer wavelength and therefore lower energy than the absorbed radiation.

Most striking when absorbed radiation is in UV and thus invisible while the emitted light is in the visible region.

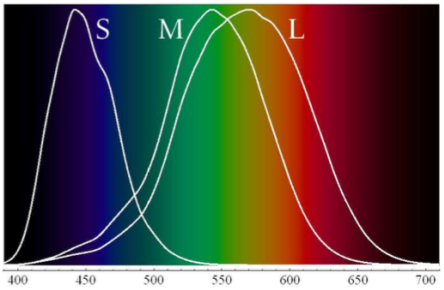


Colour Vision

We 'see' colour through the use of cones – which are the photoreceptors responsible for colour vision. They only work in daylight – when there is not enough light.

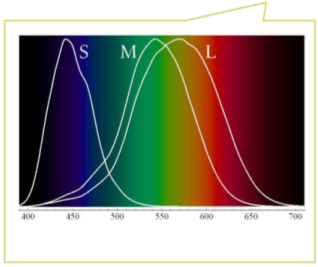
We have three types of cones:

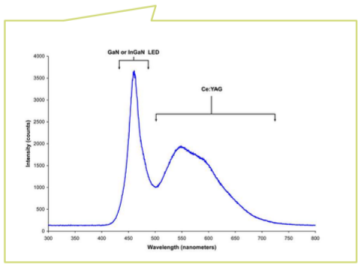
- S – sensitive to short wavelengths
- M – sensitive to medium wavelengths
- L – sensitive to long wavelengths



Sensitivity curves – probability that a photon of that wavelengths will be absorbed by a photoreceptor

- cone response = sum(sensitivity * reflected light)





Although there is an infinite number of wavelengths, we have only three photoreceptor types to sense differences between light spectra

Formally

$$R_S = \int_{380}^{730} S_S(\lambda) \cdot L(\lambda) d\lambda$$

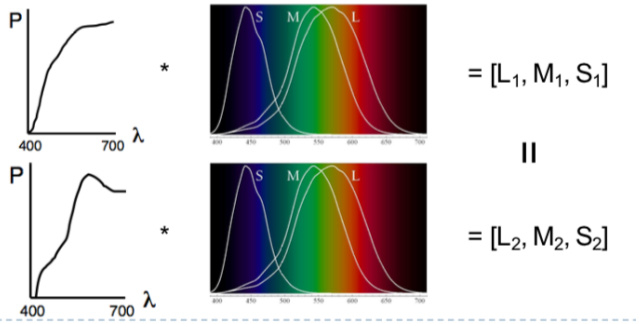
Index S for S-cones

217

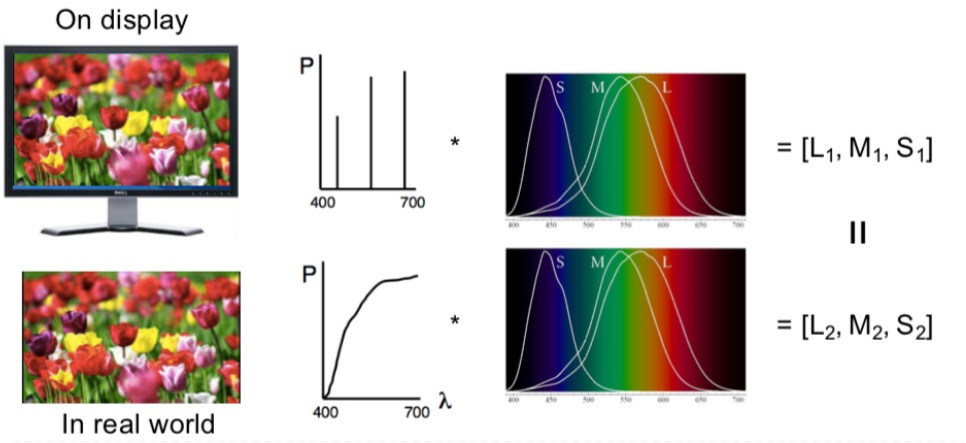
Metamerism

Due to the specific sensitivity curves, even if two light spectra are different, they may appear to have the same colour.

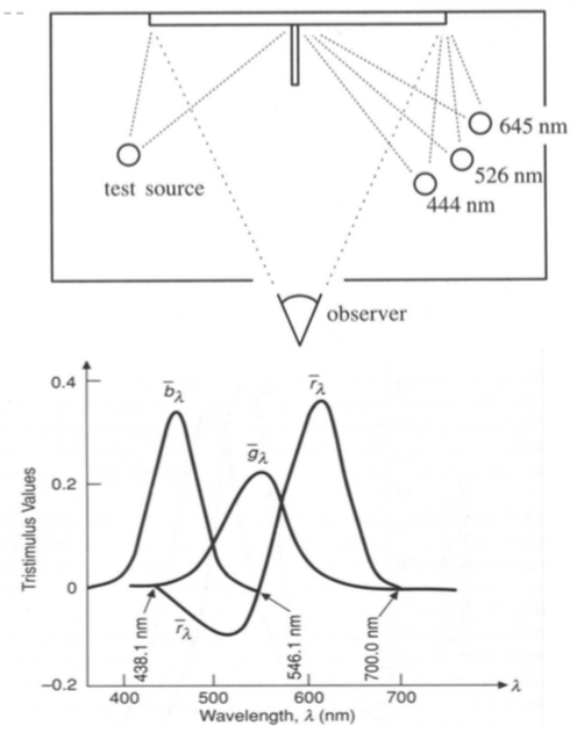
The light spectra that appear to have the same colour are called **metamers.**



We use this in the real world for displays – by displaying light in three specific wavelengths rather than continuous to replicate real world images.



It can be shown that any colour can be matched using three linear independent reference colours (**Tristimulus Colour Representation**). **However, it may require NEGATIVE contribution to test colour.** The matching curves describe the value for matching monochromatic spectral colours of equal intensity (with respect to a certain set of primary colours).

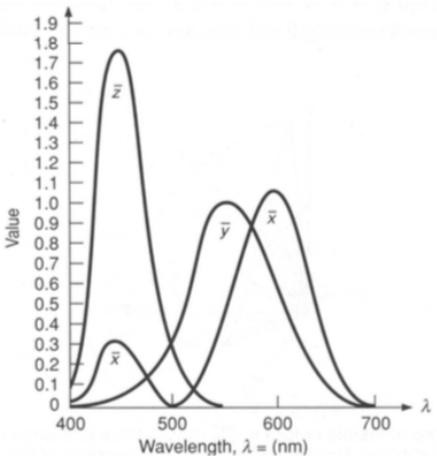


Standard Colour Space CIE-XYZ

1. CIE Experiments [Guild and Wright, 1931]
    a. Colour matching experiments
    b. Group ~12 people with normal colour vision
    c. 2-degree visual field (fovea only)
2. CIE 2006 XYZ
    a. Derived from LMS colour matching functions by Stockman & Sharpe
    b. S-cone response differs the most from CIE 1931
3. CIE-XYZ Colour Space
    a. **Goals**

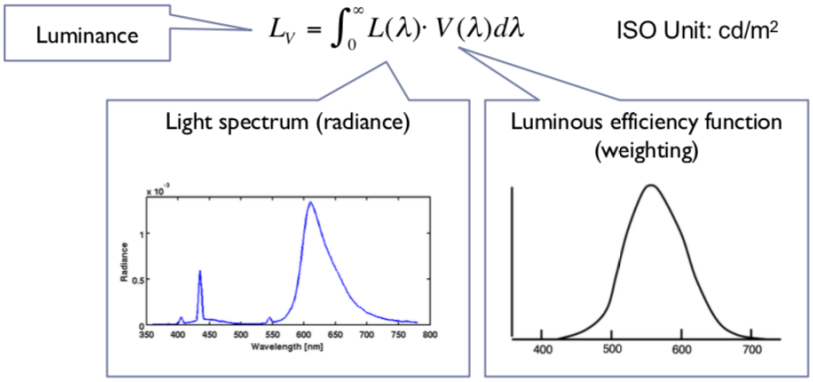
1. Abstract from concrete primaries use in experiment
2. All matching functions are positive
3. One primary is roughly proportional to light intensity

4. **Standardized Imaginary Primaries (CIE XYZ 1931)**
   a. Could match all physically realisable colour stimuli
   b. Y is roughly equivalent to luminance – the shape is similar to luminous efficiency curve
   c. Monochromatic spectral colours form a curve in 3D XYZ-space.
   d. Cone sensitivity curves can be obtained by a linear transformation of CIE XYZ
   e.



Luminance

Luminance is the perceived brightness of light, adjusted for the sensitivity of the visual system to wavelengths.



CIE Chromaticity Diagram

The chromaticity values are defined in terms of x, y, z:

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z}, \quad z = \frac{Z}{X+Y+Z} \quad \therefore \quad x+y+z=1$$

This ignores luminance and can be plotted as a 2D functions.

Pure colours (single wavelength) all lie along the outer curve and all other colours are a mix of pure colours and hence lie inside the curve. The points outside the curve do not exist as colours.

Displayable Colours
All physically possible and visible colours form a solid in XYZ space.

Each display device can reproduce a subspace of that space – a chromacity diagram is a slice taken from a 3D solid in XYZ space.

The Colour Gamut is the solid in a colour space – it is usually defined in XYZ to be device-independent.

Representing Colours

We need a mechanism to represent colour in the computer by some set of numbers. It needs to be:

- Small – so it can be quantised to be a small number of bits
  - Linear and Gamma Corrected RGB, sRGB
- Set of numbers which are easy to interpret
  - Munsell's artists' scheme
  - HSV, HLS
- Such that Euclidean colour differences are perceptually uniform
  - CIE Lab, CIE Luv

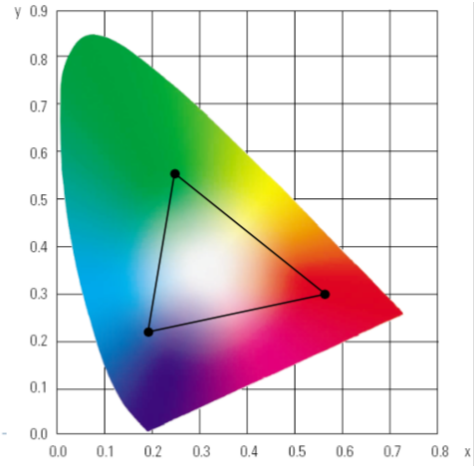
RGB Space

RGB space is cube of combining red, green and blue lights. It is used for all display devices – TVs, CRT monitors, LCDs, etc.

The device puts its own physical limitations on:

1. Range of colours
2. Brightest colours
3. Darkest colour

The red, green and blue primaries each map to a point in XYZ space and any triangle within the resulting triangle can be displayed – any colour outside the triangle cannot be displayed.



CMY Space

Printers make colours by mixing coloured inks. The important difference between inks (CMY) and lights (RGB) is that, while lights emit light and inks absorb light.

- **Cyan absorbs red, reflects blue and green**

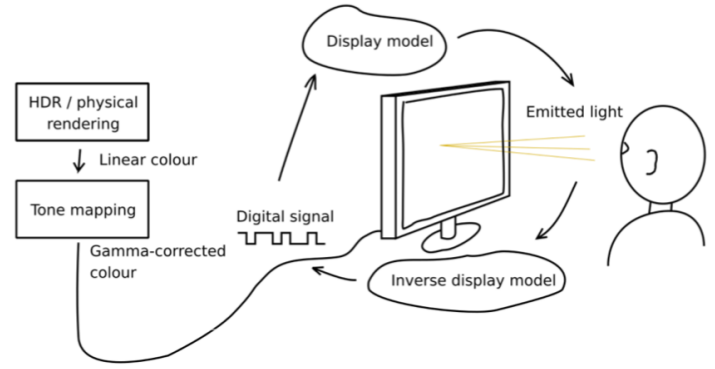
- **Magenta absorbs green, reflects red and blue**
- **Yellow absorbs blue, reflects green and red.**

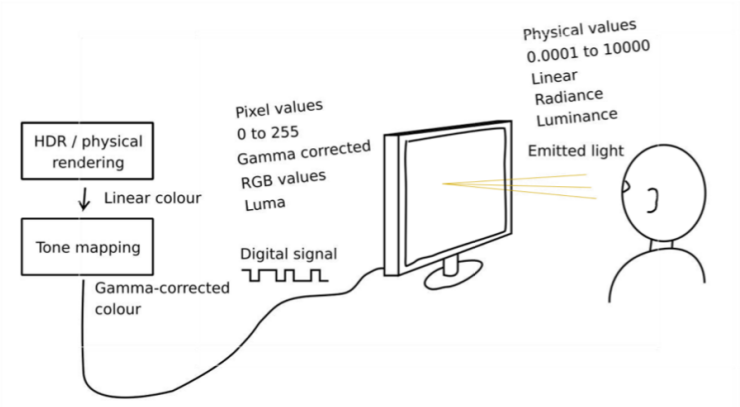
It is effectively the inverse of RGB – therefore still a cube. However, it still doesn't really work – therefore in real printing, we use black (key) as well as CMY **(CMYK Space).**

We use black because:

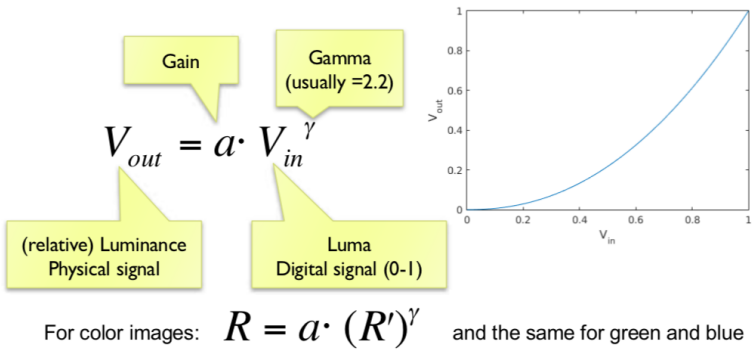
1. Inks are not perfect absorbers
2. Mixing C, M, Y gives a muddy grey, not black
3. Lots of text is printed in black: therefore, trying to align C, M and Y perfectly for black text would be a nightmare.

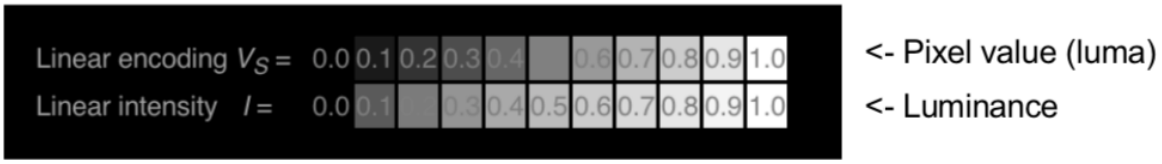
Linear vs. Gamma-Corrected Values





Gamma correction is used to encode luminance or tri-stimulus colour values (RGB) in imaging systems.





- Gamma corrected pixel values give a scale of brightness levels that is more perceptually uniform
- At least 12 bits (instead of 8) would be needed to encode each colour channel without gamma correction.
    - The response of the CRT gun

Luma

Luma is the pixel brightness in gamma corrected units:

- $L' = 0.2126R' + 0.7152G' + 0.0722B'$
- $R', G' and\ B'$ are gamma corrected colour values
- It is used in image / video coding.

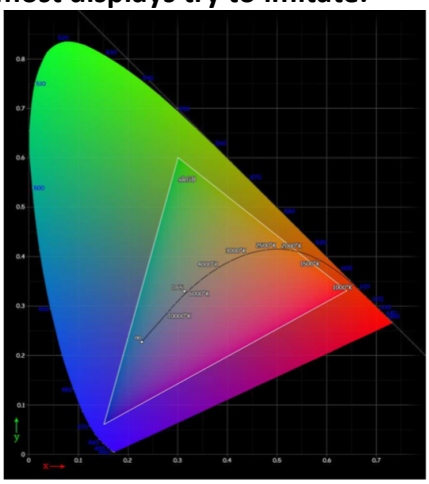
RELATIVE LUMINANCE is often approximated with:

- $L = 0.2126R + 0.7152G + 0.0722B = 0.2126(R')^{y} + 0.7152(G')^{y} + 0.0722(B')^{y}$
- Where R, G and B are linear colour values
- **Luma** and **luminace** are different quantities despite similar formulas

sRGB Space

sRGB space is not standard – colours may differ based on the choice of primaries. **sRGB is a standard colour space which most displays try to imitate:**

Chromaticity	Red	Green	Blue	White point
x	0.6400	0.3000	0.1500	0.3127
y	0.3300	0.6000	0.0600	0.3290
z	0.0300	0.1000	0.7900	0.3583



To get from XYZ to sRGB:

1. Linear Colour Transform:

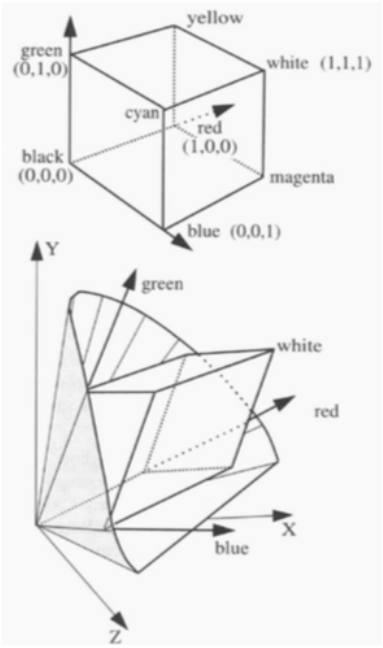
$$\begin{bmatrix} R_{\text{linear}} \\ G_{\text{linear}} \\ B_{\text{linear}} \end{bmatrix} = \begin{bmatrix} 3.2406 & -1.5372 & -0.4986 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

    a.
2. Non-linearity

$$C_{\text{srgb}} = \begin{cases} 12.92C_{\text{linear}}, & C_{\text{linear}} \leq 0.0031308 \\ (1+a)C_{\text{linear}}^{1/2.4} - a, & C_{\text{linear}} > 0.0031308 \end{cases}$$

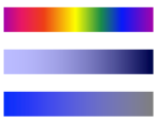
a. $a = 0.055$

b.

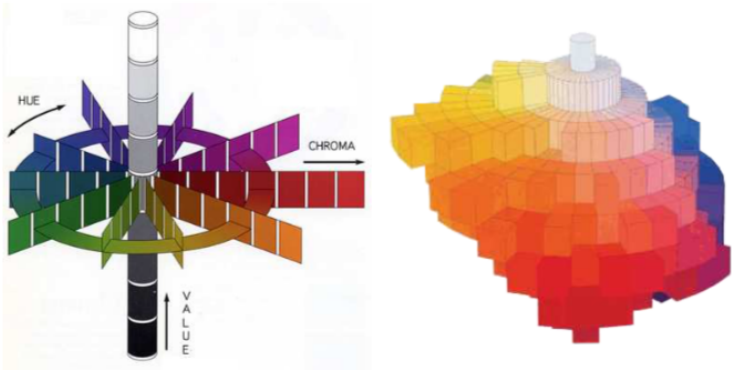


Munsell's Colour Classification System

- Invented by Albert H. Munsell, an American artist in 1905, in an attempt to systematically classify colours.
- Has three axes
  - hue ➢ the dominant colour
  - value ➢ bright colours/dark colours
  - chroma ➢ vivid colours/dull colours



- It can be represented using a 3D graph



- Any two adjacent colours are a standard 'perceptual' distance apart
  - It is worked out by testing it out on people
  - It is a highly irregular space
    - Vivid yellow is much brighter than vivid blue

Colour spaces conclusion

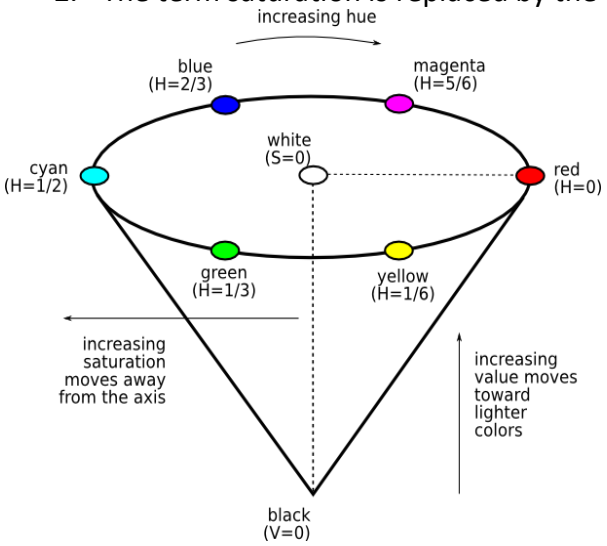
- RGB and CMY are based on the physical devices to produce the coloured output
  - They are difficult for humans to use to select colours

- Munsell's colour system is much more intuitive
    - Hue – what is the principal colour?
    - Value – how light or dark is it?
    - Chroma – how vivid or dull is it?
- Therefore, produced basic transformations of RGB which resemble Munsell's human friendly system.

HSV

Also has three axes (defined the same as Munsell's):

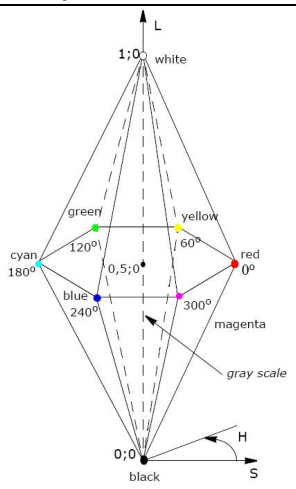
1. Hue and value have the same meaning
2. The term saturation is replaced by the term chroma



It was designed by Alvy Ray Smith in 1978 with a precise algorithm to convert between HSV and RGB.

HLS

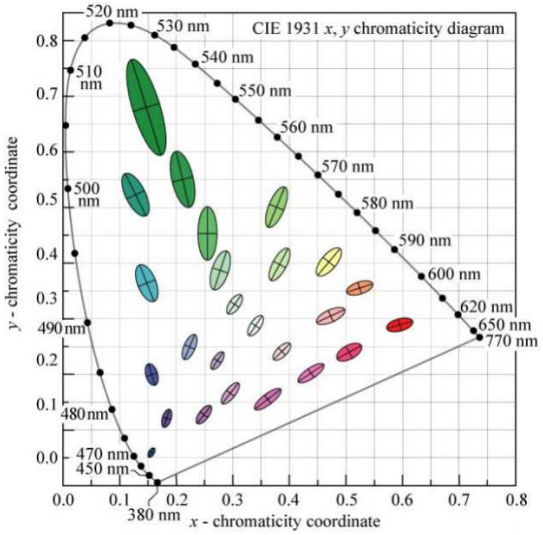
- HLS is a simple variation of HSV where hue and saturation have the same meaning and the term **lightness replace the term value.**
- It is designed to address the complaint that **HSV has all pure colours having the same lightness / value as white.**



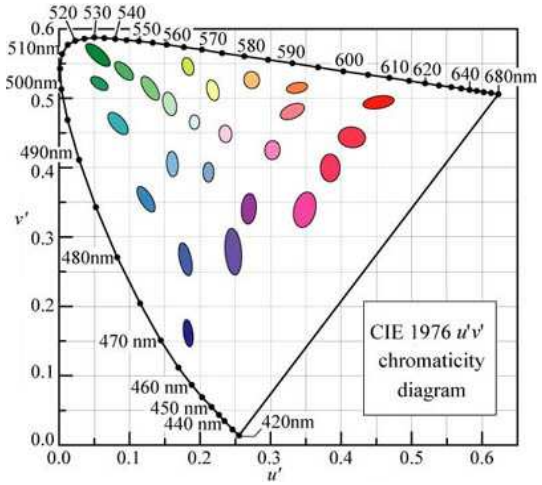
H = Hue
S = Saturation
L = Lightness

It was designed by Metrick in 1979 and as with HSV an algorithm is provided to convert between HLS and RGB.

However, in CIE xy chromatic colours and any derived things (RGB, HLS, HSV), the colours aren't perceptually uniform. You can show this by using graphs with MacAdam ellipses which groups together visually indistinguishable colours:



In CIE xy chromatic coordinates



In CIE u'v' chromatic coordinates

CIE L*u*v* and u'v'

This was made in order to be approximately perceptually uniform.

- u'v' chromacity

$$u' = \frac{4X}{X+15Y+3Z} = \frac{4x}{-2x+12y+3}$$
$$v' = \frac{9Y}{X+15Y+3Z} = \frac{9y}{-2x+12y+3}$$

- CIE LUV

Lightness

$$L^* = \begin{cases} \left(\frac{29}{3}\right)^3 Y/Y_n, & Y/Y_n \le \left(\frac{6}{29}\right)^3 \\ 116(Y/Y_n)^{1/3} - 16, & Y/Y_n > \left(\frac{6}{29}\right)^3 \end{cases}$$

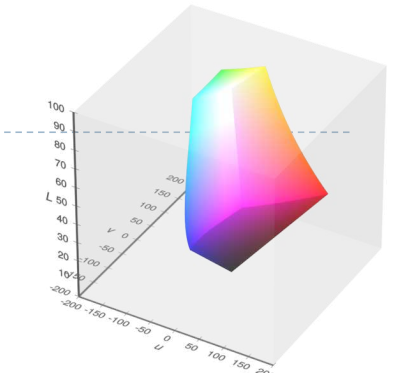
Chromacity coordinates

$$u^* = 13L^* \cdot (u' - u'_n)$$
$$v^* = 13L^* \cdot (v' - v'_n)$$

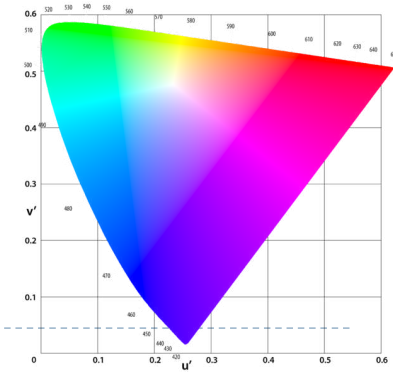
Colours less distinguishable when dark

- Hue and chroma

$$C^*_{uv} = \sqrt{(u^*)^2 + (v^*)^2}$$
$$h_{uv} = \text{atan2}(v^*, u^*),$$



sRGB in CIE L*u*v*



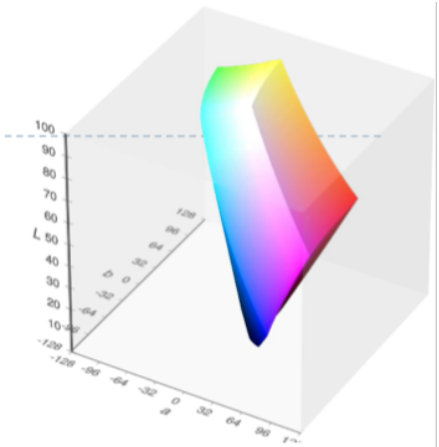
CIE L*a*b* colour space

This is another perceptually approximately uniform colour space:

$$L^* = 116 f\left(\frac{Y}{Y_\mathrm{n}}\right) - 16$$
$$a^* = 500\left(f\left(\frac{X}{X_\mathrm{n}}\right) - f\left(\frac{Y}{Y_\mathrm{n}}\right)\right)$$
$$b^* = 200\left(f\left(\frac{Y}{Y_\mathrm{n}}\right) - f\left(\frac{Z}{Z_\mathrm{n}}\right)\right)$$

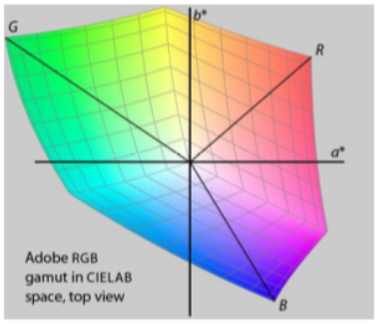
Trichromatic values of the white point, e.g.
$X_\mathrm{n} = 95.047,$
$Y_\mathrm{n} = 100.000,$
$Z_\mathrm{n} = 108.883$

$$f(t) = \begin{cases} \sqrt[3]{t} & \text{if } t > \delta^3 \\ \frac{t}{3\delta^2} + \frac{4}{29} & \text{otherwise} \end{cases}$$
$$\delta = \frac{6}{29}$$



- Chroma and hue

$$C^* = \sqrt{a^{*2} + b^{*2}}, \qquad h^\circ = \arctan\left(\frac{b^*}{a^*}\right)$$



Lab space

Lab space is a space which contains all the colours that a human can perceive. The image shows a visualisation of this. We note:

1. Human perception of colour is not uniform
2. Perception of colour diminishes at the white and black ends of the Lightness axis.
3. The maximum perceivable chroma differs for different hues.

Machine Learning and Real-World Data

Statistical Classification

Introduction to Machine Learning

- Machine Learning involve the following three aspects
    - 1) Task
    - 2) Data
    - 3) Algorithm
- Therefore, we consider
    - 1) Data Acquisition and Preparation
    - 2) Feature Extraction
    - 3) Evaluation
- **Task**
    - An abstraction from a real problem, or a piece of a larger architecture
        - In research, the large architecture is often hypothetical.
    - End-user systems often very different from experimental systems
        - Generally, research concerns standard tasks
        - Which don't really matter much in the real world
        - Reason why lots of high profile uses games – this is a task which is very easy to define.
        - Sometimes, subtle change in task makes a huge difference to research progress
            - Became easier to make progress in statistical machine translation (SMT) when the task was redefined as producing the closes match to a reference translation as measured by the BLEU score
- **Data**
    - Split into three parts
        - Training Set
        - Development (also known as the validation or tuning set)
        - Evaluation (also known as test set)
    - Acquiring Data
        - For a language processing, the data will be text (a corpus), and could be scraped from the web.
        - Can also have additional materials, such as translations into another language (parallel text)
        - For supervised learning, the available labelled data is usually quite limited – the data must be split to be used in the first three phases (Training, Development, Testing)
            - Split because we want to have new data for testing and tuning on different sets of data
            - Avoid overtraining

    - How to pick and pre-process data
        - What type of data?
        - Where should data come from?
        - How much data?
        - Is annotation needed for training or evaluation?
        - What sort of pre-processing?
- **Supervision**
    - Can have varied degrees of supervision
        - Supervised – training data is labelled with the desired outcome
            - Can be annotated manually or using a found annotation (star ratings for movie reviews).
        - Unsupervised
        - Semi-supervised
- **Features**
    - Once data has been acquired, features are extracted by the algorithm
    - External resources may be used
- **Algorithm**
    - An algorithm should be chosen which is appropriate for the task, given the available data and features.
    - Fast and dumb may be better than slow and sophisticated – especially if large amounts of data are available.
    - For any practical application, robustness to unexpected data and consistency across datasets may be more important than obtaining the highest score on evaluation dataset.
- **Evaluation**
    - Allows one to see how well a chosen algorithm performs on a given task
    - Many different metrics and approaches for different tasks
    - Evaluation may be done using humans, but for standard tasks, there are standardised metrics and test sets
    - Important to note that the best performance will depend on the details of the task

Introduction to Sentiment Classification

- We are using reviews from IMDb (Internet Movie Data Base) which has about 4.7 million titles. They have reviews which are written in natural language by the general public.
- **Sentiment Classification –** Task of automatically deciding whether a review is positive or negative based on the text – this is a standard task in NLP (Natural Language Processing)
- One possible method is using data about individual words to find the sentiment
    - **Using a lexicon** which lists over 8000 words as positive or negative
    - **Hypothesis:** A review that contains more positive than negative words is overall positive.

- It is clear that the system is not perfect, therefore important to find out how accurate we are.
- We can find out that whether it is actually positive or negative based on the star rating
  - $Accuracy = \frac{numberCorrect}{numberCorrect+numberIncorrect}$
- Accuracy of 0.635
  - Using first word to break ties - 0.6356
  - Ignoring 'weakly positive' and 'weakly negative' words – 0.664
  - Using 'weakly positive' and 'weakly negative' as half as good as 'strongly negative' and 'strongly positive' – 0.643
- Problems with using a lexicon
  - It is built using human intuition
  - Requires many hours of human labour to build
  - Is it limited to the words the humans decided to include
  - It is static – words could have different meanings in different demographics

- **Tokenization**
  - Since we are looking at words, we must divide the text into words – splitting on whitespace is often not enough
    - Words with upper case
    - Punctuation
  - **Type vs Token**
    - A token is any word, repeated or not
      - Could say that it is a word in a specific context, but can also just be a word, depending on how simplified our model is.
    - A type is a different word to those seen before
    - *The old horse went to stand next to the old man has 11 tokens but only 8 types*

Naïve Bayes Parameter Estimation

- **What is Machine Learning?**
  - A program that learns from data
  - Adapts after having been exposed to new data
  - Learns implicitly
  - Without explicit programming
- **Feature**
  - Easily observable (and not necessarily obviously meaningful) properties of the data
  - In this case, words of the review
- **Classes**
  - Meaningful labels associated with the data

  - Positive or Negative in our case
- Classification is a function that maps from features to a target class
  - A function mapping from the words in the review to a sentiment is what we are looking for.
- Probabilistic Classifiers
  - Given a set of input features, a probabilistic classifier returns the probability of each class.
  - Therefore, our prediction, is the highest probability one, given the features.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|O)$$

- Naïve Bayes Classifier
  - Liner Classifier
    - Uses a linear combination of the inputs.
  - Generative Classifier
    - They build a model of each class
    - Given an observation, they return the class most likely to have generated the observation
    - **Discriminative Classifiers are the other type**
      - They instead learn what features from the input are most useful to discriminate between the different possible classes
  - Based on Bayes' Theorem
    - Literally is basically just Bayes' Theorem – the trick is removing P(O)
  - $P(c|O) = \frac{P(c)\,P(O|c)}{P(O)}$ $where\ O\ are\ the\ set\ of\ Observed\ Features$
  - $predicted\ class\ (c_{NB}) = \ argmax_{c \epsilon C} \frac{P(c)P(O|c)}{P(O)}$
    - Argmax is the one with the maximum posterior probability
  - We can remove P(O) because it will be constant during a given classification and therefore will not affect the result of argmax
  - Therefore:
    - $predicted\ class\ (c_{NB}) = argmax_{c \epsilon C} P(c)P(O|c)$
    - P(c) is the prior probability and P(O|c) is the likelihood of the Observed features
  - For us P(O|c) = P(w1, w2, …, wn | c)
    - W1 is a word
  - **We assume there is strong independence assumption between the observed features – Naïve Bayes assumption**
    - P(O|c) ≅ P(w1|c) x P(w2|c) x … x P(wn | c)
    - So,

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i=1}^{n} P(w_i|c)$$

  - In the **training phase,** we find whatever information is needed to calculate P(wi|c) and P(c)
  - In the **testing phase,** we apply the formula to find the classifiers decision
  - **This is Supervised ML, since you use information about the classes during training**
- **Testing vs Training**
  - Testing: the process of making observations about some known data set
  - Testing: the process of applying the knowledge from training into some, new unseen data
- Implementing Naïve Bayes
  - Getting word probabilities

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$$

    - Where count (wi, c) is the number of times wi occurs with class c and V is the vocabulary of all words
  - Getting class probabilities

$$\hat{P}(c) = \frac{N_c}{N_{rev}}$$

    - Where Nc is the number of reviews with class c and Nrev is total number of reviews
  - **This all uses Maximum Likelihood Estimation (MLE) as a method to estimate the parameters of a statistical model given observations**
  - Unknown Words
    - Ignore them is the best strategy
  - **Stop Words**
    - These are very frequent words, like 'the' and 'a' which really don't mean anything at all.
    - We can ignore these by sorting the vocabulary by frequency and getting rid of the first 10.
    - However, doesn't always work very well.
  - **USING LOGS**
    - In practise we use logs to deal with multiplying lots of very small probabilities together
      - Avoid underflow and increase speed
    - So

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}}\ logP(c) + \sum_{i=1}^{n} logP(w_i|c)$$

    - Since log is monotonically increasing, we can still simply look for argmax

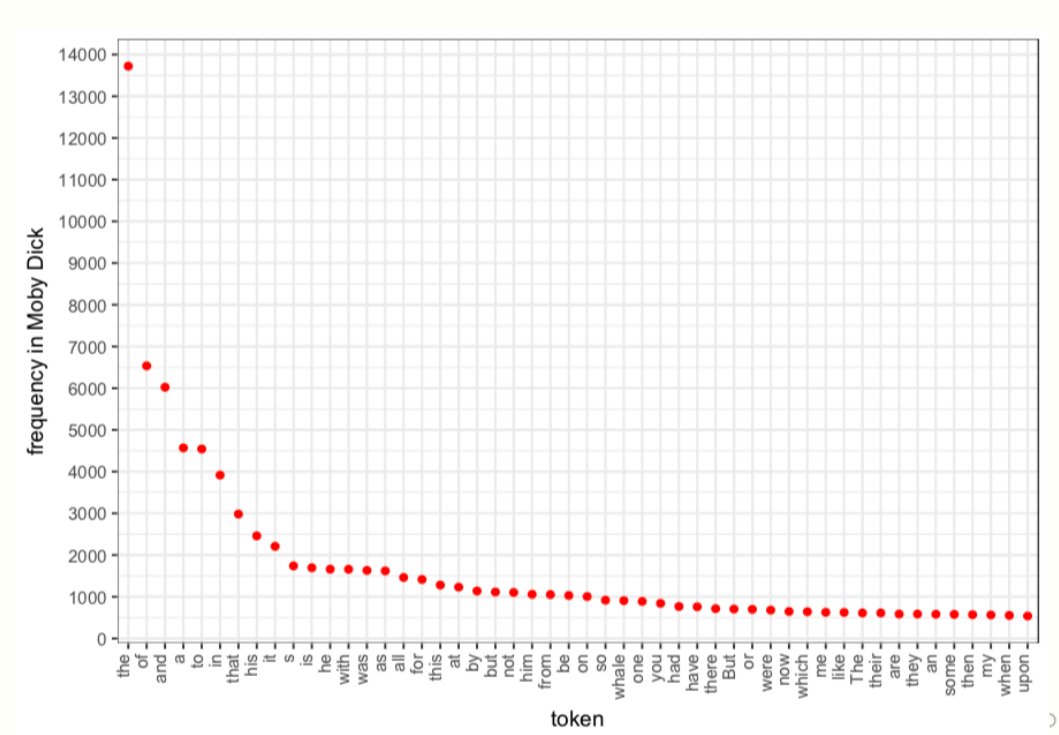
        - However, if the count is 0 for a word for a class, we will attempt to say that the probability of that word being in that class is 0 (and add negative infinity in the case of the logs), so it disappears entirely.
        - Fix this by adding smoothing:
- **Laplace (Add-One) Smoothing**

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V}(count(w, c) + 1)} = \frac{count(w_i, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|}$$

    - This fixes the problem with discovering new words
- **Binary Multinomial Bayes**
    - Same as previously, but for each review, we remove all duplicate words before concatenating them into the single big one to go through
- Further information on Naïve Bayes
    - Naïve Bayes is generally used as an effective baseline in modelling human languages
    - It clearly has assumptions which are wrong, but this is true of all formal models of human language
    - We are forced to use statistical models which are wrong because of two main things
        - Computational Tractability
        - Acquisition of Training Data
    - The Naïve Bayes assumption assumes that everything is independent from each other
        - Example of 1000 x 1000 binary pixels on a monitor (which have too many states to model exhaustively)
        - Naïve Bayes assumption is equivalent to saying that the pixels on the monitor all behave independently of each other.
        - Naïve Bayes works when we are interested in an overall characterization of some collection of sub-entities – the overall shade of the monitor – not the fact that it is displaying a cat.
    - More intelligent algorithms will take advantage of interactions between states – working with correlations which exist.

Statistical Laws of Language

- Word Frequency Distributions Power Law
    - Small number of very high-frequency words
    - Large number of low-frequency words



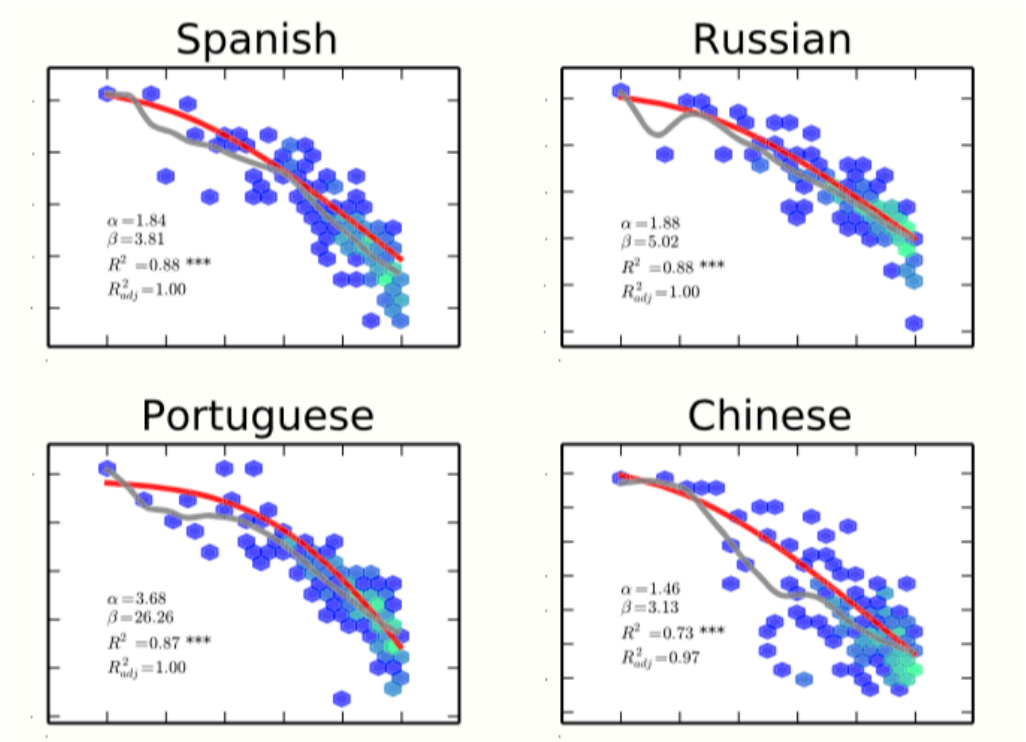
- The distribution obeys a power law – **Zipf's Law**
  - The nth most frequent word has a frequency proportional to 1/n
  - "A word's frequency in a corpus is inversely proportional to its rank"
  - The parameters of Zipf's Law are language dependent
  - $f_w \approx k/r_w^{\alpha}$
  - f_w is the frequency of word w
  - r_w is the frequency rank of the word w
  - α, k are constants (which vary with the language)
    - Alpha is around 1 for English
    - 1.3 for German
  - **Actually**
    - **Mandelbrot proposed a generalisation that more closely fits the frequency distribution in languages by shifting the range by an amount**

$$f_w \approx \frac{k}{(r_w + \beta)^{\alpha}}$$

    - $\alpha \cong 1$, $\beta \cong 2.7$
  - Zipfian distributions occur in many collections
    - Almost all languages
      - **Even in extinct and yet-untranslated languages like Meroitic**
    - Sizes of settlements

- Frequency of access to web pages
- Size of earthquakes
- Word senses per word
- Notes in musical performances
- Machine instructions
- **If a monkey randomly bangs on a typewriter, you also get this.**

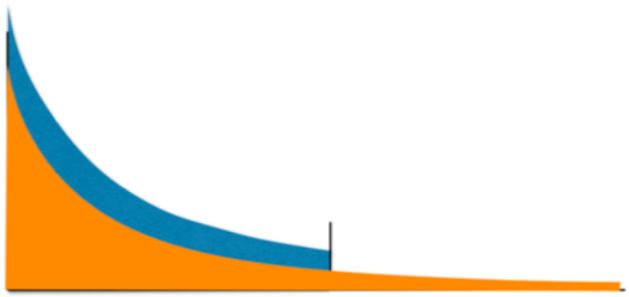
- Very useful to plot in log space



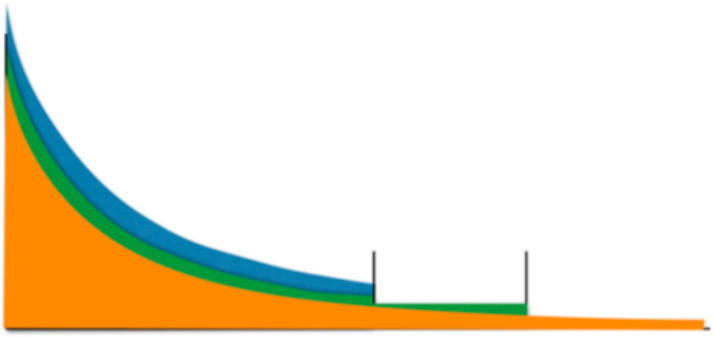
- Why does it happen?
  - No-one really knows!
    - One theory is that it can be described by communicative optimization principles
    - ‘trade-off between speakers’ and listeners’ efforts’
    - Maybe it’s just a ‘normal-esque’ distribution, given how common it is.
  - It is important to say that the apparent simplicity of the distribution of how the distribution is plotted.
  - The standard method of visualising the word frequency distribution is to count how often each word appears in a corpus and to sort the word frequency by decreasing magnitude.
  - The frequency of the rth most frequent word is then plotted against the frequency rank, typically yielding a mostly linear curve on a log-log plot.

- However, the frequency and frequency rank are estimated on the same corpus, leading to correlated errors between the x-location and y-location
- We can fix this by using two independent corpora of data to plot it.

- **Heap's Law**
  - Describes the relationship between the size of a vocabulary and the size of text that gave rise to it
  - $u_n = kn^\beta$
    - Where u_n is the number of types
    - N is the total number of tokens (text size)
    - Beta and k are constants (language-dependent)
      - Beta is around ½
      - $30 \leq k \leq 100$
  - Useful to plot in log-space
- **Effect of Zipf's Law and Heap's Law on Classifier**
  - With MLE, only seen types get a probability estimate:



  - Therefore, the estimate of the seen types is too high
  - With laplace smoothing, we redistribute the probability mass:



Statistical Tests for Classification Tasks

- Very important to be able to see which system is better and which is worse in an effective manner
  - Ensure that we do not draw conclusions and take actions based on evidence which is too weak.

  - Sometimes apparent effects may be due to natural variation in the test data
- First define a Null Hypothesis – that two result sets come from the same distribution
  - System 1 Is equally as good as system 2
- Then we choose a significance level (α) at 1% or 5% for example (0.01 or 0.05)
  - When we accept at 1% we claim that is we run the baseline system 100 times we would only once get as good performance as our new system.
- We then try to reject the null hypothesis with confidence $1 - \alpha$ $(0.99 \text{ or } 0.95 \text{ in these cases})$
- Rejecting the null hypothesis means that the observed result is very unlikely to have occurred by chance (proof beyond 'reasonable doubt').
  - We can say: "The difference between System 1 and System 2 is statistically significant at $\alpha = 0.01$"
  - Any other statements based on raw accuracy differences alone are strictly speaking meaningless.
- **Sign Test (non-parametric, paired)**
  - **The sign test uses a binary event model**
  - Here events correspond to documents
  - They have binary outcomes
    - Positive: System 1 beats System 2
    - Negative: System 2 beats System 1
    - Tie: They do equally well – this doesn't technically exist, we distribute the ties among the positive and negative.
  - Uses Binary Distribution:
    - Call the probability of a negative outcome q (here we say it is always 0.5)
      - If the null hypothesis is true, the baseline and the new system are actually equally good, but just happen to give slightly different results on particular tests.
      - Therefore, the observed counts of plus and minus give a binary distribution with a mean of 0.5n under the null hypothesis.
      - **Sign test checks how likely it is that the actual observations are a binomial distribution**
    - Probability of observing X=k negative events out of N

$$P_q(X = k|N) = \binom{N}{k} q^k (1-q)^{N-k}$$

    - At most k negative events:

$$P_q(X \leq k|N) = \sum_{i=0}^{k} \binom{N}{i} q^i (1-q)^{N-i}$$

  - If the probability of observing our events under the Null Hypothesis is very small (below α) then we can safely reject the Null hypothesis

  - **Two tailed tests**
    - So far tested difference in a specific direction
      - Probability that at most 753/2000 binary events is negative
    - However, we generally want to test for statistically significant difference regardless of direction
    - This is given by $2P(X \leq k)$ because the distribution is symmetric
  - When is it appropriate?
    - The sign test is appropriate when we have paired data, so it is a natural fit for a situation where we have tried a baseline system on some test data and want to compare a new system with the baseline.
    - We **make the assumption that the individual tests on data items are not linked**
- **Dealing with ties**
  - Split between the two
  - We go for the ceiling of the split as this makes it more unlikely that the test passes.
- **Things to learn**
  - 95% confidence = 1.96σ from the mean
  - We can check whether the observed value is more or less than two standard deviations from the mean.

Cross-Validation and Test sets

- In order to check whether a system works on all data, it is important to test the system on data that is a completely separate piece of test data.
- However, it is important to note that **overtraining** that can still happen even if we use separate test data.
  - When you think you are making improvements (performance on test data goes up) but in reality, you are making classifier worse because it generalises less well to data other than test data.
    - Indirectly also picked up accidental properties of the small test data
  - Until deployed to real unseen data, there is a danger that overtraining will go unnoticed
  - Also known as:
    - **Overfitting**
    - **Type III errors (rejecting the null hypothesis, but for the wrong reason)**
  - Ways to know if overtraining
    - Not overtraining if large amounts of test data, and use new and large enough test data each time you make an improvement
    - Not sure if incremental improvements to classifier and optimise system based on its performance on same small test data
    - You can inspect most characteristic features for each class and get suspicious when you find features unlikely to generalise
  - **Time Effects (Wayne Rooney Effect)**

        - Time changes public opinion on words / particular people
        - Therefore, important to test system on data from different time

- **Cross-Validation**
    - Why?
        1. Can't afford to get new test data every time
        2. Must never test on training set
        3. Want to use as much training material as possible
        4. Use every little bit of training data for testing
    - N-Fold Cross-Validation
        - Split data randomly into N folds
        - For each fold – use all other folds for training, test on that fold only
        - Final performance is the average of the performances for each fold.
    - Significance Test
        - Gained more usable test data and therefore are more likely to find a difference (pass the test)
    - **Stratified Cross-Validation**
        - Cross-Validation where each split is done in a way such that it mirrors the distribution of classes observed in the overall data
    - **Analysis**
        - Find average and **variance:**

$$var = \frac{1}{n}\sum_{i}^{n}(x_i - \mu)^2$$

- **How good is it?**
    - From testing – according to the textbook, the average of the cross-validation may not necessarily get the right true accuracy – with the system having a higher degree of flexibility.
    - In fact, importantly it gets an accuracy which is higher.
    - Therefore, we should only be interested in the minimum point of the error curve and trying to get that to apparently increase rather than necessarily getting the average to be higher.

Uncertainty and Human Agreement

- **Adding more classes**
    - So far only contained positive or negative reviews
    - Need to also consider neutral reviews
- **Finding true category**
    - We can use **human agreement** as a source of truth

- Something is 'true' is sever humans agree on their judgement, independently from each other
- The more they agree, the truer it is
- We can find the observed agreement:

$$P(A) = \text{MEAN}\left(\frac{\text{observed rater–rater pairs in agreement}}{\text{possible rater–rater pairs}}\right)$$

- **P(A) observed agreement**
  - Pairwise observed agreement: average ratio of observed to possible rater-rater agreements
  - There are n_C2 = ½n(n-1) possible pairwise combinations between n judges
  - P(A) is the mean of the proportion of prediction pairs which are in agreement for all items (sum up the ratios for all items and divide by the number of items)
  - **However,** we want to see how much better this is than chance
    - **The probability of two independent judges choosing a class blindly – following the observed distribution of the classes is:**

$$P(E) = P(\text{both choose POSITIVE or both choose NEGATIVE})$$
$$= P(\text{POSITIVE})^2 + P(\text{NEGATIVE})^2$$

  - **Fleiss' Kappa measures reliability of agreement**

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)}$$

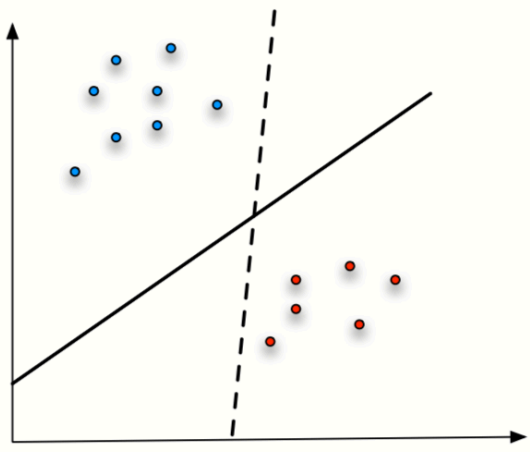
    - Degree of agreement over that which would be expected by chance
    - P(A) (Observed Agreement): average ration of observed to possible pairwise agreements
    - P(E) (Chance Agreement): sum of squares of probabilities of each category
    - P(A) – P(E) gives the agreement above chance
    - 1 – P(E) gives the agreement that is attainable above chance
    - **Analysis**
      - If k = 1 then complete agreement
      - If k = 0 then no agreement beyond chance
      - If k < 0, then observed agreement less than expected by chance
      - 0.8 indicates very good agreement
      - **But, no universally accepted interpretation**
      - Size of k is affected by the number of categories and may be misleading with a small sample size.
      - **Weighted Kappa was created that allows you to weight different degrees of disagreement.**
      - Issues in two specific ways as well

    - **1) Trait Prevalence**
        - Occurs when the characteristic being measured is not distributed uniformly across items
        - Since the P_e is vastly increased for times when there is more of one class than another.
        - The value of Fleiss' Kappa not only depends on actual agreement (and the accuracy of the system), but also the distribution of the **Gold Standard** scores: **these are the correct classes**
    - **2) Marginal Homogeneity**
        - Second problem is that the difference in the marginal probabilities (probabilities distribution of the results) affects the coefficient considerably
        - Comes from the assumption that marginal probabilities are classification propensities that are fixed.
        - Kappa could lead to learning functions that favour assigning distributions that are different to that of the gold standard.
        - **Tends to affect weighted kappa less**
- **Other Measurement Metrics**
    - Pearson's Product Moment Correlation
        - Parametric measure of association that quantifies the degree of linear dependence between two variables and describes the extent to which the variables co-vary relative to the degree to which they vary independently.
        - The greater the association, the more accurately one can use the value of one to predict the other.
        - However, outliers can destroy the coefficient.
    - Spearman's Rank Correlation Coefficient
        - Non-parametric measure of coefficient
        - Calculating by ranking the variables and computing Pearson on the ranks rather than the raw values
        - Unlike Pearson, has robustness to outliers
        - But, not really an appropriate measurement for his task, where we would like actual agreement with respect to the scores
            - Also, do not account for any systematic biases in the data.
            - High correlation can be observed in even if the predicted scores are 0.01 higher than the gold standard.
    - Scott's Pi Metric
        - Sensitive to trait prevalence but not to marginal
    - Agreement Coefficient AC and Brennan Prediger Coefficient

        - Both estimate P_e more conservatively using more plausible assumptions.
    - **Desirable Features for a Metric**
        - **1) Robustness to trait prevalence**
        - **2) Robustness to marginal homogeneity**
        - **3) Sensitivity to magnitude of misclassification**
        - **4) Robustness to score range**
    - According to the paper, the best is the AC coefficient with quadratic weights - with Cohen's Kappa generally being bad with 1, 2 and 4.
        - However, it is independent on the type of data that it is being used on, that is, whether there is a categorical or ordinal (gold standard) scale.

Introduction to other classifiers

- Naïve Bayes is a probabilistic classifier – we find the probability of features and classes based on data and therefore pick the item with the highest probability.
- **Support Vector Machine (SVM)** is a non-probabilistic binary linear classifier
    - SVMs assign new examples to one category or the other
    - SVMs can reduce the amount of labelled data required to gain good accuracy
    - A linear SVM can be considered to be a base-line for non-probabilistic approaches
    - SVMs can be efficiently adapted to perform non-linear classification
    - SVMs find hyper-planes that separate classes



    - It finds the maximum-margin hyperplane in noisy data such that distance from it to the nearest data point from each class is maximised
    - **Efficient and Effective**
        - When learning from large number of features
        - When learning from small amounts of labelled data
        - We can choose how many points to involve (size of margin) when choosing the plane (tuning vs overfitting)
        - We can separate non-linear boundaries by increasing the feature space (using a kernel functions)
- **Decision Tree**

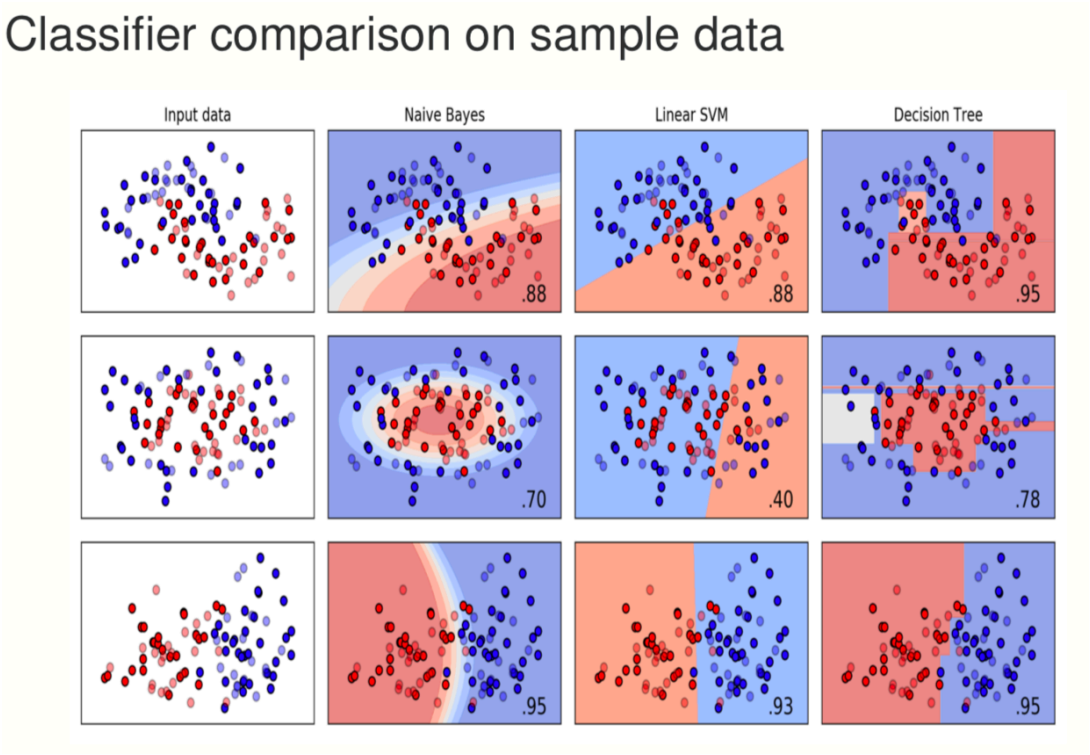
- A decision tree can be used to visually represent classifications
- It is very simple to interpret, and you can mix numerical and categorical data.
- You can specify the parameters of the tree (maximum depth, number of items and lead nodes)
- However, finding the optimal decision tree can be np-complete
- **Information Gain**
  - It is defined in terms of entropy H

$$H(n) = -\sum_{p} p_i \log_2 p_i$$

    - Where p's are the fraction of each class at node n
  - Used to decided which feature to split on at each step in building the tree

$$I(n, D) = H(n) - H(n|D)$$

    - Defines the information gain where H(n|D) is the weighted entropy of the child nodes.



Sequence Analysis

Hidden Markov Models and HMM training

- Idea is that we can use a history of observations to predict the next one.
- **Markov Assumption (first order)**
  - **The next state is ONLY based on the previous state – Limited Horizon**

$$P(w_t \mid w_{t-1}, w_{t-2}, \ldots, w_1) \approx P(w_t \mid w_{t-1})$$

- Using data, we can come up with a transition probability matrix, such as this one:

$$\begin{array}{cc} & \begin{array}{cc} \multicolumn{2}{c}{\text{Tomorrow}} \\ Rainy & Cloudy \end{array} \\ \text{Today} \begin{array}{c} Rainy \\ Cloudy \end{array} & \left[\begin{array}{cc} 0.7 & 0.3 \\ 0.3 & 0.7 \end{array}\right] \end{array}$$

- **Markov Chain is a stochastic process that embodies the Markov Assumption**
  - Can be viewed as a probabilistic finite-state automaton.
  - The states are fully observable, finite and discrete; the transitions are labelled with transition probabilities.
  - It models sequential problems – your current situation depends on what happened in the past.
  - **It is useful for modelling the probability of a sequence of events that can be unambiguously observed**
- **Hidden Markov Model**
  - **Nothing more than a probabilistic function of a Markov process**
  - **Assumptions**
    - Limited Horizon – Markov Assumption
    - Time Invariant
      - Probability of something happening at one time is equal to it happening at any time.
    - Output independence
      - Probability of an output observation depends only on the state that produced the observation and not on any other states or any other observations
  - There are hidden states, which we can't see. There is also a set of observed states, which we can.
  - But the underlying Markov Chain is over the hidden states.
  - We have no 1-1 mapping between observations and hidden states.
  - We now have to infer the sequence of hidden states that correspond to a sequence of observations.
  - **We tend to create a fast state and a final state (with no observations)**
    - It can in many ways be thought of as a non-deterministic finite state automaton with probabilities attached to each arc.
    - The Markov properties ensure that we have a finite state automaton.
    - There are no long-distance dependencies, and where one ends up next depends simply on what state one is in.
  - **Definitions:**

$$\begin{aligned} S_e = \{s_1, \ldots, s_N\} \quad & \text{a set of } N \text{ emitting hidden states,} \\ s_0 \quad & \text{a special start state,} \\ s_f \quad & \text{a special end state.} \end{aligned}$$

$K = \{k_1, \ldots k_M\}$ an output alphabet of M observations ("vocabulary").
k_0 a special start symbol,
k_f a special end symbol.

$O = O_1 \ldots O_T$ a sequence of T observations, each one drawn from K.

$X = X_1 \ldots X_T$ a sequence of T states, each one drawn from S_e.

- **Assumptions**
  - MARKOV ASSUMPTION: Transitions depend only on the current state
  - OUTPUT INDEPENDENCE: Probability of an output observation depends only on the current state and not on any other states or any other observations
- **Implementation**
  - Create a state transition probability matrix of size (N+2) x (N+2) where N is the number of states
    - A_{ij} is the probability of moving from state s_i to state s_j
  - Create an emission probability matrix of size (M + 2) x (N + 2) where M is the number of observations
    - $B_i(k_j)$ is the probability of emitting vocabulary item k_j from state s_i
  - Our hidden Markov Model is defined by its parameters μ = (State Transition Probability Matrix, Emission Probability Matrix)
  - **Problem 1 is Labelled Learning – getting two matrices**
  - **Problem 2 is Unlabelled Learning – learning using only the observation sequence and the set of emitting states**
  - **Problem 3 is Likelihood – given a HMM and an observation sequence, find out its likelihood**
  - **Problem 4 is Decoding – given an observation sequence and a HMM, find the most likely hidden state sequence**
- **Applications**
  - Speech Recognition
    - Observations: audio signal
    - States: phonemes
  - Part-of-speech tagging (Noun, Verb, etc)
    - Observations: words
    - States: part-of-speech tags
- **Why us a HMM?**
  - Very useful when one can think of underlying events probabilistically.
  - One of a class of models for which there are efficient methods of training through using **Expectation Maximization algorithm**

        - Algorithm lets us automatically learn the model parameters that best account for the observed data.
    - We can use HMMs to generate parameters for linear interpolation of n-gram models.
        - Use the hidden states being whether to use unigram, bigram, trigram, etc probabilities
        - The HMM will determine the optimal weight to give to the arcs entering each of these hidden states, which represents the amount of the probability mass which should be determined by each n-gon model, by setting their parameters.
- **Arc-Emission HMM vs State Emission HMM**
    - **Arc-Emission HMM:** The symbol emitted at time t depends on both the state at time t and at time t+1
    - **State Emission HMM:** Symbol emitted at time t depends only on the state at time t.
- **Fully Connected HMM (Ergodic)**
    - HMM where there is a non-zero probability of transitioning between any two states.
- **Bakis HMM (left-to-right)**
    - HMM where there are no transitions between a higher numbered state to a lower numbered state – only transitions going upwards.
- **Fundamental Questions for HMMs**
    - **1) Likelihood:** Given a model μ = (A, B), how do we efficiently compute how likely a certain observation is, that is P(O|μ) (**Forward Algorithm)**
    - **2) Decoding:** Given the observation sequence O, and a model μ, how do we choose a state sequence that best explains the observations **(Viterbi)**
    - **3) Learning:** Given the observation sequence O, and a space of possible models found by varying the model parameters, how do we find the model that best explains the observed data?

The Viterbi Algorithm

- **Decoding:** finding the most likely hidden sequence X that explains the observation O given the HMM parameters μ

$$\begin{aligned}\hat{X} &= \underset{X}{\operatorname{argmax}} P(X, O|\mu) \\ &= \underset{X}{\operatorname{argmax}} P(O|X, \mu) P(X|\mu) \\ &= \underset{X_1 \ldots X_T}{\operatorname{argmax}} \prod_{t=1}^{T} P(O_t|X_t) P(X_t|X_{t-1})\end{aligned}$$

    - **Viterbi Algorithm**

- The Viterbi algorithm helps us to accomplish this in an efficient manner, using Dynamic Programming
- We can use Dynamic Programming if
  - There is an optimal substructure property
    - An optimal state sequence X1, … Xj, … Xt contains within in the sequence X1 … Xj which is also optimal
  - Overlapping Subsolutions Property
    - If both Xt and Xu are on the optimal path, with u > t, then the calculation of the probability for being in Xt is part of each of the many calculations for being in state Xu
- Do the calculation of the probability of reaching each stage once for each time step
- Then memoise this probability in a Dynamic Programming table – **trellis**
  - (N + 2) x (T + 2) with states j as rows and time steps as columns
  - Each cell records the Viterbi Probability of the most likely path that ends at that time (being in that state, having made the correct observations):

$$\delta_j(t) = \max_{1 \le i \le N} [\delta_i(t-1)\, a_{ij}\, b_j(O_t)]$$

    - This probability is calculated by maximising over the best ways of going to s_j for each s_i.
    - a_{ij}: the transition probability from s_i to s_j
    - $b_j(O_t)$: the probability of emitting O_t from destination state s_j

    δ(t-1) – The **previous Viterbi Path Probability** from the previous time step
    A_{ij} – **Transition probability**
    $B_j(o_t)$ – **State observation likelihood**

  - You should set the probability of starting the sequence at t=0 as 1 for start state and 0 for everything else.
  - It is also useful to have a table that will help you to back trace and find the actual most likely path, which should simply tell you which state came last.
- This reduces our effort to $O(N^2T)$

function VITERBI(*observations* of len T, *state-graph* of len N) **returns** *best-path*

create a path probability matrix *viterbi*$[N+2,T]$
for each state s **from** 1 **to** N **do** ; initialization step
  $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
  $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
return the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

- We can also give a formal definition of the Viterbi recursion as follows:

1. **Initialization:**

$$v_1(j) = a_{0j} b_j(o_1) \quad 1 \le j \le N$$
$$bt_1(j) = 0$$

2. **Recursion** (recall that states 0 and q_F are non-emitting):

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$
$$bt_t(j) = \operatorname{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$

3. **Termination:**

$$\text{The best score:} \quad P* = v_T(q_F) = \max_{i=1}^{N} v_T(i) * a_{iF}$$
$$\text{The start of backtrace:} \quad q_T* = bt_T(q_F) = \operatorname{argmax}_{i=1}^{N} v_T(i) * a_{iF}$$

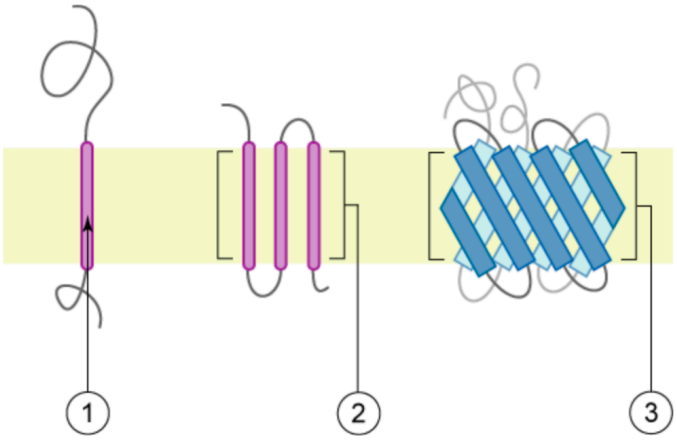
- **Precision and Recall**
  - The human labels we define as correct are called **gold labels**
  - **So, therefore, we have true positives, false positive, false negatives and false positives**

- We have accuracy – ratio of observations the system labelled correctly.
- This doesn't work very well when the test data is not very well distributed
  - When we have 900 and 100 of another class, if we simply randomly predict everything is in the first class, then we get an accuracy of 0.9, even though the system is clearly terrible
- So, we define precision and recall
  - $Precision = \frac{true\ positives}{true\ positives + false\ positives}$
  - $Recall = \frac{true\ positives}{true\ positives + false\ negatives}$
- We combine the two metrics, precision and recall into a single metric called **F-measure**
  - $\frac{(\beta^2+1)PR}{\beta^2 P+R}$
  - We set β depending on what we want to weight more heavily
  - B > 1 favours recall
  - B = 1 precision and recall are equally balanced
  - B < 1 favours precision
  - We generally often use F_1 measure
    - $F_1 = \frac{2PR}{P+R}$
  - F-measure comes from a weighted harmonic mean of precision and recall.
    - The harmonic mean of a set of numbers is the reciprocal of the arithmetic mean of reciprocals.
    - It is used because it is a conservative metric – the harmonic mean of two values is closer to the minimum of the two values than the arithmetic mean is.

Using an HMM in a biological application

- Cell membranes are lipid bilayers: inside the membrane is hydrophobic, the two sides are hydrophilic
  - They have let things in an out of the cell
  - Proteins which are part of the cell membrane allow this
- In cell metabolism: proteins make sure the right thing happens in the right place at the right time. The proteins are made of amino acid sequences – with each amino acid having very different side chains.
- The sequences fold into very complex 3D structures.
- **Transmembrane Proteins**
  - These are proteins which go through the membrane one or more times
  - The regions of the protein which lie inside and outside the cell tend to have more hydrophilic amino acids.
  - The regions inside the membranes tend to have more hydrophobic amino acids
  - Many of them proteins involve alpha-helixes in the membrane

- The channel formed by the proteins allows ions to pass through in a controlled way



1. a single transmembrane α-helix (bitopic membrane protein)
2. a polytopic transmembrane α-helical protein 3. a polytopic transmembrane β-sheet protein

- We can image and get an accurate structure using x-ray crystallography, but it is difficult and time-consuming, and we don't know the membrane location

- **HMM**
  - **Hidden Sequence is inside the membrane or inside or outside the cell**
  - **Observed sequence is amino acid**
  - This is much quicker and easier than x-ray crystallography
  - It helps us distinguish the interior of the membrane from the inside / outside of the cell

Monte Carlo Methods

- **Protein Folding**
  - **Anfinsen's Hypothesis:** The structure a protein forms in nature is the global minimum of the free energy and is determined by the amino acid sequence
  - **Levinthals's Paradox:** Protein folding takes milliseconds – not enough time to explore the space and find the global minimum. Therefore, a kinetic function must be important.
  - **Finding Protein Structure:** The Primary structure may be determined directly or from DNA sequencing
    - **The Secondary and Tertiary structure** can be determined by x-ray crystallography and other direct methods, but difficult, time-consuming
    - We want to be able to predict the structure given the amino acid sequence
    - Secondary structure prediction is relatively tractable, various prediction methods, including HMMs
    - But, tertiary structure prediction is very difficult
- **Protein Tertiary Structure Prediction**

    - Modelling fully is hugely computationally intensive
    - Methods:
        - Molecular Dynamics (MD): modelling chemistry: use home computers to run simulations.
        - Foldit: Get lots of humans to work on the problem (an example of gamification)
        - **Use Monte Carlo Methods (repeated random sampling to explore possibilities)**
- **Monte Carlo methods in protein structure prediction**
    - Objective: find lowest energy state of protein
    - Idea
        - Start with secondary structure
        - Try pseudo(random) move – see if the result is lower energy and repeat
    - Problem
        - **Local Minima – locally good move might not be part of best solution**
            - So also sometimes accept a move that increases energy
    - Specific Approach
        - **Metropolis-Hastings – a type of Markov Chain Monte Carlo method**
- **Monte Carlo methods in general**
    - Using random sampling to solve intractable numerical problems
    - Buffon's needle for estimating pi
    - Physicists developed modern Monte Carlo methods at Los Alamos: programmed into ENIAC by von Neumann
    - Bayesian statistical inference did not come until 1993 – this is essential for many modern machine learning approaches

Social Networks

Properties of Networks

- What are we looking for?
    - What role does a node (entity / person) play in this network?
    - Is there a substructure?
        - Neighbourhood areas / cliques
- Uses
    - Academic investigation of human behaviour (sociology, economics)
    - Disease transmission in epidemics
    - Modelling information flow – who sees a picture, reads a piece of news (or fake news)
    - Identifying links between sub-communities
        - Erdös Number

        - Steps in a path between Paul Erdos and a researcher, counting co-authorship of papers as links
    - Finding countries with powerful positions in an international trade graph
    - Split online journalism into clusters
        - Turns out it splits well into liberal and conservative
        - Then can pick the most used pages
    - **Networks in the natural world**
        - Food webs
            - Allows us to consider cascades of removing nodes with cascading extinctions
        - Brain structure
        - Metabolism networks
- **Degree:** Number of neighbours a node has in a graph
    - The distribution of node degrees may be very skewed
- **Distance**
    - Length of shortest path between two nodes
- **Cycle**
    - Informally a ring structure
    - Formally, a path with at least three edges in which the first and last nodes are the same, but otherwise all nodes are distinct.
- **Diameter**
    - Maximum distance between any pair of nodes
    - **Small World Phenomenon: Six Degrees of Separation**
        - Chain Links was a short story by Karinthy in 1929 (book says it was a play by John Guare), which claimed that any two individuals in the world could be connected via at most 5 personal acquaintances.
        - Hard to prove, but has been shown in settings where we do have full data available on the network structure – Erdos Number, Bacon Number (very hard to find people with numbers above 6!)
            - However, have been some experiments on a small scale which have showed this idea working.
        - All comes down to very small distances to the Giant Component
- Natural networks tend to have closely clustered regions connected only by a few links between them – these are often week ties.
- **Giant Component:** A connected component containing a significant proportion of the nodes in a graph
    - This is a deliberately vague term
    - ‘When a network contains a giant component, it almost always contains only one’
        - Unlikely to have two coexisting giant components
        - At some point, they would get joined.
- **Weak and Strong Ties:** The closeness of a link – how many connections

    - Strong ties tend to be embedded in tightly-linked regions of the network
    - Weak ties, representing more casual connections, tend to cross these regions
        - More crucial to the structure
    - We can think about social networks in terms of the strongly tied networks and the weak connections that connect them.
- **Structural Balance:** How local breaks in connections can affect a network
- **Structural Holes:** Empty space in the network between two sets of nodes that do not otherwise interact closely.
    - Purposefully vague definition
- **Connectivity**: Graph is connected if, for every pair of nodes, there is a path between them
    - If a graph is not connected, then it breaks apart naturally into a set of pieces – components.
    - **Connected Component:** Subset of nodes such that every node in the subset has a path to each other and the subset is not part of some larger set which is connected.
- **Bridge:** Edge connecting two components which would otherwise be unconnected (removing the edge would make one component into two)
    - Bridges are generally very rare in real social networks
- **Local Bridge:** An edge joining two nodes that have no other neighbours in common. Cutting a local bridge increases the length of the shortest path between the nodes.
    - **Span of a local bridge:** The distance the two nodes would be if we remove the node.
        - No local bridge of span 2
- **Triadic Closure:** Triangle of nodes.
    - **Thought of as a dynamic property: if A knows B and A knows C, relatively likely B and C will get to know each other**
    - A, B and C are a triangle in the network
    - **Reasons for Triadic Closure**
        - In human systems
            - Large opportunity to meet them
            - And there is a basis for trust
            - “incentive for the common node to bring them together as it will be a source of latent stress’
- **Strong Triadic Closure Property**
    - Triadic closure is especially likely to occur if the two connections are strong ties.
    - “Node A violates the Strong Triadic Closure property if it has strong or weak ties to two other nodes B and C, and there is no edge at all (strong or weak) between B and C”
    - If a node A in a network satisfies the Strong Triadic Closure property and is involved in at least two strong ties, then any local bridge it is involved in must be a weak tie.
- **Clustering Coefficient:** A measure of the amount of triadic closure in a network

  - Probability that two randomly selected neighbours of a node are also neighbours of each other.
  - More strongly the triadic closure process operates, the higher the coefficient
  - **Relevance in human networks**
    - Very high correlation between having a low clustering coefficient and high rates of suicide in a group
- **Braess's Paradox**
  - Idea that interactions among people's behaviour can lead to counterintuitive effects
  - Adding resources to a transportation network can create incentives which undermine efficiency
- **Embeddedness:** The number of common neighbours shared between two endpoints
- **Neighbourhood Overlap**
  - Defined for an edge from A to B
  - $$\frac{number\ of\ nodes\ who\ are\ neighbours\ of\ both\ A\ and\ B}{number\ of\ neighbours\ who\ are\ neighbours\ of\ at\ least\ one\ of\ A\ and\ B}$$
- **Random Networks**
  - Is the small world phenomenon surprising?
    - Exponential number of connections means we *should* reach other people very quickly
    - Since we have triadic closure, this limits the number of paths you have to take to get to someone nearby
    - It is clear that long weak ties are crucial
  - Watt and Strogatz: Randomly generated graph with triangles at close range plus a few long random links
    - **Mathematically Defined**
    - Describes a graph $G_{n, k, p}$ = (V, E) where n is |V|, k is the initial degree of a node (an even integer) and p is a rewiring probability
    - It starts with a ring of nodes, with each node connected to its k nearest neighbours with undirected edges – **it is a ring lattice**
    - Each of the starting edges (u, v) is visited in turn (u < v, starting with a circuit of the edges to the nearest neighbours, then to the nest nearest and so on) and replaced with a new edge (u, v') with probability p, with v' chosen randomly.
    - If p is 1, every edge is rewired randomly and is p is 0, there is no rewiring.
    - Watt and Strogatz also say that n >> k >> ln(n) >> 1 where k >> ln(n) guarantees that the graph will be connected.
  - Erdós-Renyi Model
    - Random undirected graph generation where $G_{n, p}$ = (V, E) where n is the number of vertices |V| and the probability of forming an edge (u, v) is given by p

        - A variance is to generate all the possible graphs with the given number of vertices and edges and select between them with uniform probability, but this is not much used.
- **Implementation of Task**
    - **How to find diameter of the network**
        - BFS from any one point and find the node it finds last
        - Then BFS from that node and find the node it finds last (and the distance) – that is the diameter
        - **Alternatively**
            - Breadth-first all-pairs shortest path – O ($V^3 + EV^2$)

Betweeness Centrality

- **Aim:** Find gatekeeper nodes via the betweeness centrality
    - Certain nodes / edges are most crucial in lining densely connected regions of the graph
    - Cutting these edges isolates the cliques / clusters
- **Local Bridge:** An edge which increased the shortest paths if cut
- **Nodes which are intuitively the gatekeepers can be determined by the betweeness centrality.**
- **Betweeness Centrality:**
    - Proportion of shortest paths between all pairs of nodes that go through the node
    - **Naïve Approach**
        - For every node, use a BFS to find all the shortest path between s and all the nodes. Store all these paths for each pair s, t
        - For each vertex, count the number of shortest paths which go through the node.
            - Given a directed graph G = <V, E>
            - σ (s, t) is the number of shortest paths between nodes s and t
            - σ (s, t | v) is the number of shortest paths that pass-through v
            - c_b (v) is the betweeness centrality of v
                - $c_b(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$
                - If s = t, then σ(s, t) = 1
                - If v ϵ s, t then σ(s, t | v) = 0
            - σ(s, t) can be calculated recursively

$$\sigma(s,t) = \sum_{u \in Pred(t)} \sigma(s,u)$$

$Pred(t) = \{u : (u,t) \in E, d(s,t) = d(s,u) + 1\}$
predecessors of t on shortest path from s
$d(s,u)$: Distance between nodes s and u

Therefore, we can run a Breadth First Search from each node as a source once, for a total complexity of $O(V^2 + EV)$

- The naïve algorithm uses a lot of space
    - We shouldn't really need to save anything about the paths between s and t once we've updated $C_B(v)$ for each vertex v on those paths
    - So, we could (Storage Efficient Algorithm)

1. **For every node v in V, set $C_b(w) = 0$**
2. **For each node s in V**
    - **Use a BFS algorithm to find all the shortest paths between s and all other nodes. Store the paths for each target t**
    - **For each t, for each vertex that occurs on one of the stored paths, count the number of times w appears in total, and divide by the total number of paths between s and t.**
    - **Add this result to $C_b(w)$**
3. **$C_b(w)$ gives the final result**

- Integration with Shortest Paths Algorithm
    - Easier to add things going backwards, rather than checking if every node is on the path
    - For every node v, set $C_b(v) = 0$
    - For every node s in V
        - Set S(v, t) to zero for all nodes v and t in V
        - Use the BFS algorithm, as above, to reach target t from
        - In the backward phase, increment S(u, t) as appropriate when each node v is reached, rather than creating full paths
        - At the end, divide each S(u, t) by the total number of paths between s and t
        - Add the result to $C_b(v)$
    - $C_b(w)$ gives the final result

- Recursive method
    - The main difference between what we have above and Brandes' efficient algorithm is that the latter make use of a recursive step on the backwards phase to allow direct calculations of the ration for each v on the basis of its successor nodes on the shortest paths to every t.
    - For now, we assume we have such a function, and a value δ(v) for which the following hold

- δ(t) = 0 if t is a terminal node (no nodes below them on shortest paths)
- We can increment δ(v) via this function every time we reach v from a node w on the backwards phase
- After we have finished with all the w values, δ(v) can be straightforwardly be accumulated into Cb(v)

1. For every node v in V, set $C_B(w)$ = 0.

2. For each node s in V:

    1. set $\delta(v)$ to zero for all nodes v in V.
    2. Use the BFS algorithm (much as before, differences in bold below) while Q is not empty, do:
        1. dequeue v from Q and push v onto a stack S
        2. For each node w such that is an edge in E from v to w, do
            1. if dist[w] is infinity, then
               set dist[w] to dist[v] + 1
               enqueue w
            2. if dist[w] = dist[v]+1 then
               set $\sigma(s, w)$ to $\sigma(s, w) + \sigma(s, v)$
               append v to Pred[w]
    3. while S is not empty, pop w off S
        1. for all nodes v in Pred(w) set $\delta(v)$ to $\delta(v) + \text{MAGIC}(\delta(w))$.
        2. unless $w = s$, set $C_B(w) = C_B(w) + \delta(w)$.

3. $C_B(v)$ gives the final result.

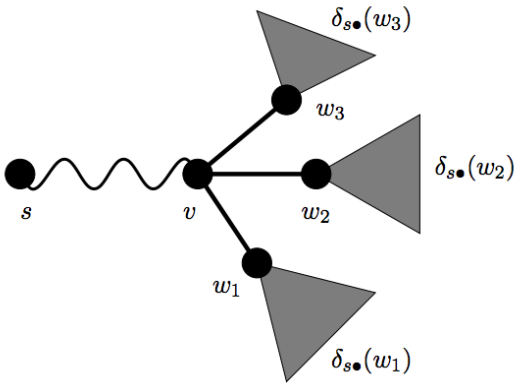
    - It turns out this function requires us to know the number of shortest paths between s and each node v so we create these values in the forward phase of the BFS as shown.
    - We also need to make sure that the nodes are visited in the correct order on the backward step, which we do with the stack
- Brandes' Algorithm
    - Ratio between the shortest paths between s and t that go through v and the total number of shortest paths between s and t is:
        - $$\delta(s, t|v) = \frac{\sigma(s, t|v)}{\sigma(s, t)}$$
    - Therefore:
        - $$C_B(v) = \sum_{s,t \in V} \delta(s, t|v)$$
    - If we define **one-sided dependencies as:**

$$\delta(s|v) = \sum_{t \in V} \delta(s, t|v)$$

- And:

$$C_B(v) = \sum_{s \in V} \delta(s|v)$$

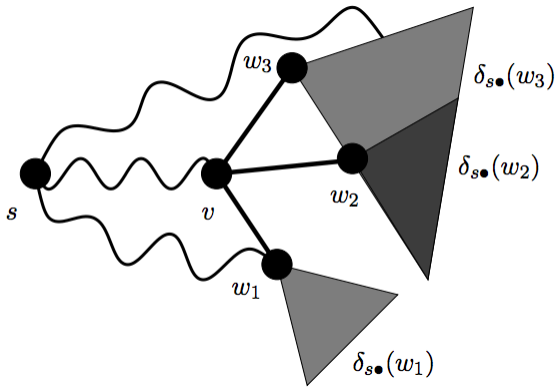
- **Assuming only one shortest path (tree condition)**
  - If the vertices and edges of all the shortest paths form a tree, it is clear that δ(v) can be computed simply



  - In a bit more detail
    - For any vertex v and any target t, either v lies on the unique shortest path between s and t, in which case δ(s, t|v) = 1, or it doesn't, δ(s, t|v) = 0
    - For any vertex w, such that v immediately precedes on shortest paths from s, v will lie on the shortest path from s to w
    - So, for any node x, st w immediately precede x on shortest paths, v will lie on the shortest paths.
    - Therefore, we can total all the one-sided dependencies:

$$\delta(v) = \sum_{w} (1 + \delta(w))$$

- **Multiple shortest paths**
  - We need to find a way to determine the ration of shortest paths which will go through v



- Then,

$$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v) = \sum_{t \in V} \sum_{w\,:\,v \in P_s(w)} \delta_{st}(v, \{v, w\}) = \sum_{w\,:\,v \in P_s(w)} \sum_{t \in V} \delta_{st}(v, \{v, w\}).$$

-
- Let w be any vertex with v ϵ $P_s(w)$. Of the σ_{sw} shortest paths from s to w, many first go from s to v and then use {v, w}
- Therefore,
  - $\frac{\sigma_{sv}}{\sigma_{sw}} \cdot \sigma_{st}(w)$
  - Shortest paths from s to some t contain v and {v, w}.
  - It follows that the pair dependency of s and t on v and {v, w} is:

$$\delta_{st}(v, \{v, w\}) = \begin{cases} \frac{\sigma_{sv}}{\sigma_{sw}} & \text{if } t = w \\ \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \frac{\sigma_{st}(w)}{\sigma_{st}} & \text{if } t \neq w \end{cases}$$

- Putting this into the above equation gives:

$$\begin{aligned} \sum_{w\,:\,v \in P_s(w)} \sum_{t \in V} \delta_{st}(v, \{v, w\}) &= \sum_{w\,:\,v \in P_s(w)} \left(\frac{\sigma_{sv}}{\sigma_{sw}} + \sum_{t \in V \setminus \{w\}} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \frac{\sigma_{st}(w)}{\sigma_{st}} \right) \\ &= \sum_{w\,:\,v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w)) \,. \end{aligned}$$

- **Intuition**
  - Several cases to consider depending on whether the bypassing edge arrives at one of the nodes w or beyond it
  - The intuition is that only the situation where the bypassing edge arrives at one of the w that we have to worry about.
  - Otherwise, δs of the ws already incorporate the ratios of the shortest paths accordingly.
  - Therefore, only need to look at the ratio of paths going between s and v compared to those going between s and w to capture the extent to which there are bypassing paths going to w.

- **Brandes' Algorithm**
  - *Intended for directed graph, but works the same for undirected graph, except that each pair is considered twice, once in each direction*
  - *Therefore, halve the scores at the end for undirected graphs.*
    - Iterate over all vertices s in V
    - Calculate δ(s | v) for all V

- Breadth-first search, calculating distances and shortest path counts from s, push all vertices onto stack as they're visited.
- Visit all vertices in reverse order (pop off stack) aggregating dependencies according to the equation.

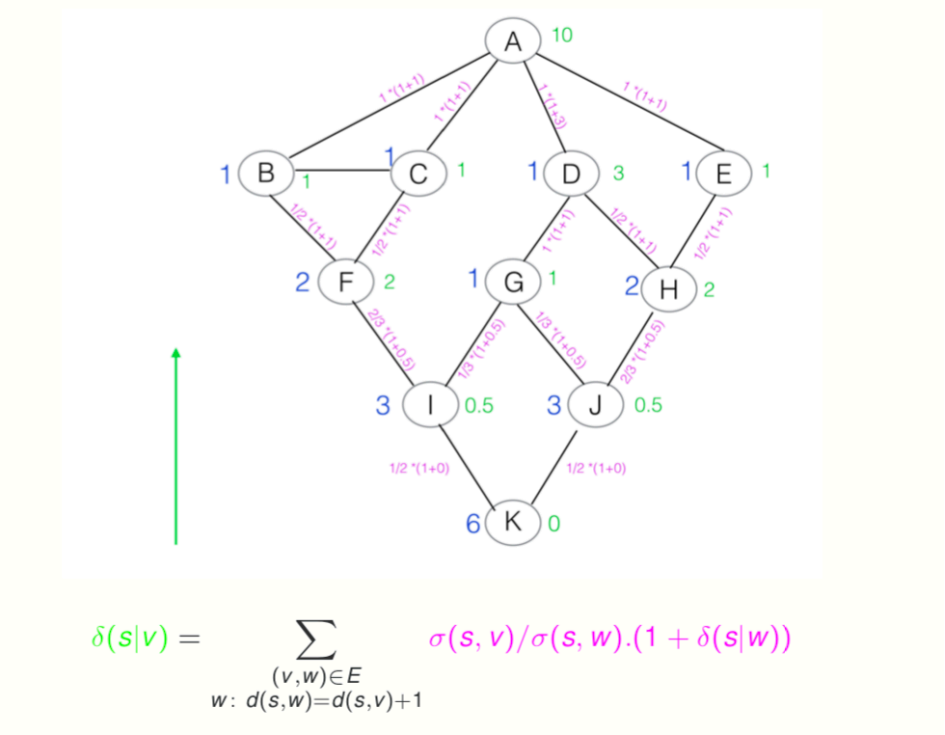
input: directed graph $G = (V, E)$
data: queue Q, stack S (both initially empty)
and for all $v \in V$:
$dist[v]$: distance from source
$Pred[v]$: list of predecessors on shortest paths from source
$\sigma[v]$: number of shortest paths from source to $v \in V$
$\delta[v]$: dependency of source on $v \in V$
output: betweenness $c_B[v]$ for all $v \in V$ (initialized to 0)

for $s \in V$ **do**
▼ **single-source shortest-paths problem**
▼ **initialization**
for $w \in V$ **do** $Pred[w] \leftarrow$ empty list
for $t \in V$ **do** $dist[t] \leftarrow \infty$; $\sigma[t] \leftarrow 0$
$dist[s] \leftarrow 0$; $\sigma[s] \leftarrow 1$
enqueue $s \rightarrow Q$
while Q *not empty* **do**
dequeue $v \leftarrow Q$; push $v \rightarrow S$
foreach *vertex* w *such that* $(v, w) \in E$ **do**
▼ **path discovery** // — w found for the first time?
if $dist[w] = \infty$ **then**
$dist[w] \leftarrow dist[v] + 1$
enqueue $w \rightarrow Q$
▼ **path counting** // — edge (v, w) on a shortest path?
if $dist[w] = dist[v] + 1$ **then**
$\sigma[w] \leftarrow \sigma[w] + \sigma[v]$
append $v \rightarrow Pred[w]$
▼ **accumulation** // — back-propagation of dependencies
for $v \in V$ **do** $\delta[v] \leftarrow 0$
while S *not empty* **do**
pop $w \leftarrow S$
for $v \in Pred[w]$ **do** $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w])$
if $w \neq s$ **then** $c_B[w] \leftarrow c_B[w] + \delta[w]$

- **Single-Source Shortest Path Algorithm**
  - Given a source, effectively breadth first search, keeping track of the distances
  - Every time you see something new, add it to the queue and set its distance

- If you see a node which will keep shortest path algorithm – distance to that is distance to current node + 1, add one to the shortest paths going through this edge.
- Add this node to the list of predecessors for that node.
- Also, add this to the stack when you process a node, so you can do the back propagation of dependencies

- **Accumulation for one start!**



$$\delta(s|v) = \sum_{\substack{(v,w)\in E \\ w:\ d(s,w)=d(s,v)+1}} \sigma(s,v)/\sigma(s,w).(1+\delta(s|w))$$

- Go backwards through the nodes, and simply apply the formula

- **Accumulation**
  - Perform the V iterations, once with each node as the source and sum up the δ(s|v) for each node, giving the node's betweeness centrality.

Clustering using betweeness centrality

- **Clustering:** Automatically grouping data according to some notion of closeness or similarity
  - **Agglomerative Clustering** works bottom up
    - "glue regions together"
  - **Divisive Clustering** works top-down by splitting the graph
- **Newman-Girvan method**
  - Form of divisive clustering
  - Criterion for breaking links is **edge betweeness centrality**
  - When do we know when to stop breaking links?
    - Use prior knowledge when to stop based on numbers of clusters
    - Inherent 'goodness of clustering' metric – using modularity

while number of connected subgraphs $<$ specified number of clusters (and there are still edges):

1. calculate edge betweenness for every edge in the graph
2. remove edge(s) with highest betweenness
3. recalculate number of connected components

Note:

- Treatment of tied edges: either remove all (today) or choose one randomly.

- **Determining Connected Components**
  - Depth-First search, start at an arbitrary node and mark the other nodes you reach
  - Repeat with unvisited nodes until all are visited
- **Alternative, Manual Method**
  - Supplement the network with numerical estimates of tie strength for the edges, based on empirical evidence
  - Delete edges of minimum total strength
- **Edge Betweeness Centrality**
  - With Node betweeness, we measure the number of shortest paths between two nodes going through a particular node.
  - This time we are measuring the number going through a particular edge
  - The algorithm only changes in the bottom-up (accumulation) phase.
  - $\delta(v)$ is as before, but $c_b(v, w)$

```
for s ∈ V do
    ▼ single-source shortest-paths problem
        ▼ initialization
            for w ∈ V do Pred[w] ← empty list
            for t ∈ V do dist[t] ← ∞;   σ[t] ← 0
            dist[s] ← 0;   σ[s] ← 1
            enqueue s → Q
        while Q not empty do
            dequeue v ← Q;   push v → S
            foreach vertex w such that (v, w) ∈ E do
                ▼ path discovery  // — w found for the first time?
                    if dist[w] = ∞ then
                        dist[w] ← dist[v] + 1
                        enqueue w → Q
                ▼ path counting  // — edge (v, w) on a shortest path?
                    if dist[w] = dist[v] + 1 then
                        σ[w] ← σ[w] + σ[v]
                        append v → Pred[w]
    output: betweenness c_B[q] for q ∈ V ∪ E (initialized to 0)
    ▼ accumulation
        for v ∈ V do δ[v] ← 0
        while S not empty do
            pop w ← S
            for v ∈ Pred[w] do
                c ← (σ[v]/σ[w]) · (1 + δ[w])
                c_B[(v, w)] ← c_B[(v, w)] + c
                δ[v] ← δ[v] + c
            if w ≠ s then c_B[w] ← c_B[w] + δ[w]
```

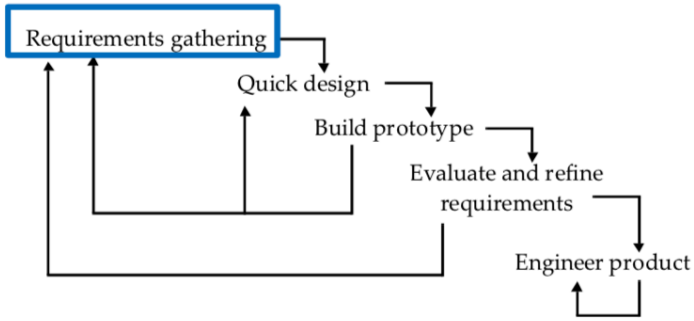
- **Facebook Circles Dataset**
  - Designed to allow experimentation with automatic discovery of circles
    - Facebook friends from a particular social group
  - Profile and network data from 10 Facebook ego-networks
    - Networks emanating from one person are referred to as an ego
  - Gold-standard circles – manually identified by the egos themselves
  - Average of 19 circles per ego
  - Complete network has 4,039 nodes in 193 circles
  - When doesn't Newman-Girvan work?
    - Nodes may be in multiple circles
    - Sub-categories within categories

        - Nodes not in any category
- **Evaluating Simple Clustering**
    - We assume data sets with gold standard or ground truth clusters
    - But we do not generally have labels for clusters, the number of clusters found may not equal true classes
    - **Purity:** Assign label corresponding to majority class found in each cluster, then count correct assignments, divide by total elements

Interaction Design

Requirements Analysis & Development

Idea of interaction design is **iterative user-centred design and development**



Requirements arise from understanding the users' needs – establishing what is needed and what the system should accomplish.

Why?

1. Things can go wrong
    a. Ambiguous specification from customer
    b. Designer misinterpretation
2. Define requirements is where failure most often occurs
3. Failure at this stage is expensive and time-consuming to fix

What?

- **Requirement:** What or how software should perform
- **Aims:**
    - (1) Identifying Needs: Understand as much as possible about users, task, context
    - (2) Establishing Requirements: Produce a stable set of requirements
- **Activities (Iterative):**
    - Data gathering
    - Data analysis
    - Defining requirements

Types

1. Functional Requirements: What the system should do
2. Non-functional Requirements: What are constraints on the system
    a. Data requirements
        i. Where is it coming from?
        ii. What kinds of data?
        iii. How will they be stored?
        iv. How long will it persist?
        v. How accurate will the stored data be?
        vi. Which data representations are needed?
    b. Environmental requirements
        i. Physical
        ii. Social
        iii. Organisational

   iv. Technical
c. User characteristics
   i. Demographics
   ii. ICT related abilities
   iii. Physical and mental disabilities
   iv. Preferences
d. Usability goals and user experience goals
   i. Usability: effectiveness, efficiency, safety, utility, learnability, memorability
   ii. Experience: enjoyable, entertaining, aesthetically pleasing, motivating

How?

1. Socio-technical models
   a. CUSTOM Stakeholder Analysis – identifying the stakeholder (anyone who is affected by the success or failure if the system).
      i. Stakeholder definition
         1. Those who interact directly with the product
         2. Those who manage direct users
         3. Those who receive output from the product
         4. Those who make the purchasing decision
         5. Those who user competitor's products
         6. **Primary –** people who use the system (frequent, hands-on)
         7. **Secondary –** people who produce input for the system, or receive output from the system
         8. **Tertiary –** people who are affected by the system's introduction, or will influence its purchase
         9. **Facilitating –** people involved in the system's design, development and maintenance
   b. Requirements Development – understand the stakeholders
      i. Aims – what does the stakeholder have to achieve, and how is success measured
      ii. Sources of satisfaction – sources of satisfaction, dissatisfaction and stress for a stakeholder
      iii. Knowledge and skills – what knowledge and skills does the stakeholder have?
      iv. Attitudes to work – what is the stakeholder's attitude towards work and computer technology?
      v. Work-group attributes – any attitudes that affects the acceptability of the product to the stakeholder?
      vi. Nature of activities – characteristics of the stakeholder's task in terms of (1) frequency, (2) fragmentation, (3) choice of actions
      vii. Responsibility – any specific issues related to (1) responsibility, (2) security, (3) privacy
      viii. Working conditions – typical conditions in which the stakeholder is working?
2. Soft Systems Methodology (SSM)

a. Considering the organisation as a whole – stakeholders and technology are components of the larger context. The learning and appreciation of the problem situation rather than in order to solve a problem.
b. Three elements:
    i. (1) Rich picture – detailed description of the problem situation
        1. Who are stakeholders?
        2. What groups?
        3. What tasks?
        4. Using
            a. Interviews
            b. Observation of work practises
            c. Interactive approaches
    ii. (2) Root definitions – stakeholder perceptions
        1. Moves focus of analysis from the real-world situation to definitions of what stakeholders perceive the activities that are taking place
        2. CATWOE
            a. Clients – people who benefit or accept output from system
            b. Actors – stakeholders who perform activities
            c. Transformations – what changes lead to things changing in the environment
            d. World View – how the system is perceived by a client
            e. Owner – who the system belongs to
            f. Environment – what factors influence the system
    iii. (3) Conceptual Model – constructed with the details of what the system has to do to meet the root definitions
        1. Using transformation – (1) what is achieved and (2) how. They are listed hierarchically.

Gathering and Analysing Data

Aim: Collect (1) sufficient, (2) relevant and (3) appropriate data – stable set of requirements.

Data Collection Techniques

1. Observation
    a. Direct Observation in the field – seeing how problem is currently solved – whilst talking to the users
    b. Indirect Observation – trying to emulate the same process – asking user to talk about what they are doing.
    c. While clearly realistic settings, it is difficult to set up and intrusive.
2. Interviews
    a. (1) Unstructured (open questions), (2) Structured (closed questions), (3) Semi-structured (open and closed questions)
    b. Useful to identify user's subjective opinions
    c. Deciding questions, things to avoid:
        i. Long questions
        ii. Compound questions
        iii. Complex jargon
        iv. Leading questions

        v. Unconscious biases
    d. Can also use props
3. Focus Groups
    a. Used to identify conflicts in terminology or expectation from different groups.
    b. Need to select participants to represent well the target users
4. Card Sorting
    a. Sorting cards into groups – interactive process
    b. Provides:
        i. Terminology
        ii. Relationship
        iii. Categories
    c. Three types:
        i. Open Card Sorting – no pre-stablished groupings
        ii. Closed card sorting – there is a pre-established grouping
        iii. Hybrid – some combination of the two
5. Questionnaires
    a. In order to elicit some specific information
    b. Getting answers to specific questions from large dispersed group of people
    c. Qualitative or quantitative data – use with other techniques
    d. Types of questions:
        i. Open Questions – respondent free to write whatever they want
        ii. Closed Questions – respondent selects answers from a set of presented possibilities
            1. Simple checklist
            2. Multi-point rating scale
            3. Ranked order
    e. Things to remember
        i. (1) Question order important
        ii. (2) Need different versions for different populations
        iii. (3) Clear instructions needed
        iv. (4) Avoid long questionnaires – or have shorter version
        v. (5) Decide whether phrases all positive, all negative or mixed.
        vi. (6) Clear purpose of study
        vii. (7) Anonymity is important
        viii. (8) Incentive
        ix. (9) 40% response rate good, 20% acceptable
6. Studying Documentation
    a. Good source of data about steps (and regulations) written down in manuals
7. Researching similar products
8. Web analytics
    a. Focus on number of web visitors and page views

Choosing Techniques:

1. Time
2. Detail
3. Risk
4. Required knowledge

5. Kind of task
6. Need to involve all stakeholder groups – more than one representative from each
7. Support process with prototypes and task descriptions
8. Balance functional and non-functional requirements

Data Analysis and Interpretation

1. Quantitative Analysis
    a. Numerical methods – averages, percentages
    b. Card Sorting analysis
        i. Looking for commonalities of groups
        ii. For small projects simply looking works
        iii. For larger projects – **cluster analysis**
            1. Similarity Rating – add up times two cards appear together and divide by number of groups
2. Qualitative Analysis
    a. Expresses nature of elements
    b. Represented as themes, patterns, stories
    c. Looking for critical incidents
    d. **Theoretical Frameworks –** basing data analysis around theoretical frameworks provides further insight
        i. Grounded Theory
        ii. Distributed Cognition
        iii. Activity Theory
        iv. Thematic Analysis

Design Process and Prototyping

Aim: How to optimise the user's interaction such that it supports and extends the user's activities in a useful, efficient and usable way.

Participatory Design (User-Centred Design):

Has the idea that the users are experts on their work situation – has three ideas:

1. Work Focussed
    a. Design concentrates on improving the workers' environment, rather than requirements
2. Collaborative
    a. Designers and users collaborate at every step

design → **measure** (against the requirements) → **test** (with users) → **redesign**

Conceptual Design – abstractly describes the system's intended behaviour
Physical Design – addresses specific, concrete layout and design issues
Interactive prototypes – allow the users to interact with the designs

Pipeline:

1. Brainstorming

2. Concept development – high level description of how a system is organised and operates. It allows designers to straighten out their thinking before they start laying out their widgets.
    a. What is the driving concept or metaphor behind the design – need one concept to make a coherent design
    b. Moodboard is a type of collage consisting of images, text and samples of object in a composition of the choice of the creator.
3. Prototyping
    a. **Why?**
        i. Explore a design space through multiple iterations
        ii. Demo versions to users (get more objective feedback)
        iii. Develop designs – 'thinking through making'.
        iv. Identify the most important features
        v. Choose between alternatives
    b. **What?**
        i. Technical aspects, with workflows working and there being the screen layouts and information displays existing.
        ii. Describe the look-and-feel
    c. Low fidelity
        i. Rough designs – consideration of user use
        ii. Problems identified from trouble users have as they walk through the system
    d. Types
        i. Paper
        ii. Video
        iii. Form model
        iv. Wireframe
4. Storyboarding
    a. Shows a rough idea of user's activities – presented as a sequential storyline
    b. Helps user communicate with designers – what they do and how they do it
5. Workshops

Personas:

- Used to capture a set of user characteristics – not real people but synthesised from real users.
- Brought to life with a name, characteristics, goals, personal background

Workshops:

- Provide a forum for discussion where designers and users can ask each other about their perspectives.
- Generally used to fill in gaps of understanding about the situation

Task Analysis and Modelling

What? Hierarchical composition of knowledge – HTA (Hierarchical Task Analysis).
Why? Understand how people currently perform work – and use it to inform design. The system will fail if it (1) doesn't do what the users want and (2) is inappropriate for the user.

How?

- First identify (1) goals, then (2) the actions to meet the goals and (3) the sequential dependencies.

Task Model

- Task Decomposition (how is this task done)
    - Decompose the high-level tasks and break them down into their constituent subtasks.
    - At a lower level show the task flows, decision processes and screen layouts.
- Show sequencing from left to right
- Consistency
    - Checking tasks are broken down to the same level of sub-tasks.
- Issues
    - Does not scale well
    - Does not allow for
        - Overlapping tasks
        - Interruptions
        - Learning
        - Communication
    - Tends to concentrate on how things are already done

Task Flow Diagrams

Documents the details of specific tasks (serves to identify problems) – not only show the specific details of current work processes but may also highlight areas:

- Where task processes are poorly understood
- Where task processes are carried out differently by different staff
- Where task processes are inconsistent with higher level task structure

Principles of Good Design

We are concerned with the **usability** of the interfaces – refers to (1) how well users can learn and use a product to achieve their goals and (2) how satisfied they are with that process.

Important Questions:

1. How easily to determine the function of the interface?
2. How easily to tell what actions are possible?
3. How to determine mapping from intent to physical movement?
4. How easy to perform the action?
5. How to tell what state the system is in?

Design Principles

These are generalised abstractions for thinking about different aspects of design – they are the do's and don'ts of interaction design. They are derived from a mix of theory-based knowledge, experience and common-sense.

Common Design Principles

1. **Visibility** - be able to work everything out

2. **Feedback** – sending information back to the user about what has been done
3. **Constraints** – restricting the possible actions that can be performed, helping prevent users from selecting incorrect options
4. **Consistency** – interface to have similar operations and use similar elements for similar tasks
    a. **Internal Consistency** – operations behave the same within an application
    b. **External Consistency –** designing operations to be the same across applications and devices
5. **Affordances –** refers to an attribute of an object that allows people to know how to use them.
    a. Norman (1998) used the term to discuss the design of everyday objects – since then popularised in interaction design to discuss how to design interface objects
    b. Interfaces are better conceptualised as perceived affordances – learned conventions of arbitrary mappings between action and effect at the interface.

Shneiderman's Golden Rules for Interface Design

1. Consistency – consistency in the way the system looks and works
    a. Terminology
    b. Aesthetics – consistent colour codes, layout, fonts, across windows
    c. Symbols
    d. Response – respond to input in the same way every time
2. Universal Usability – allow frequent users to develop a clear idea of how the system works and lets them work faster – shortcuts, toolbars, hotkeys
3. Informative Feedback – for every user actions, there should be some feedback from the systems. For frequent and minor actions, the response can be modest, but for major actions, response should be more substantial.
4. Dialogs with closure – design interactions should have a beginning, middle and end
5. Prevent errors – design system so users cannot make a serious error
    a. If they do make an error, the system must be able to detect it and offer easy to understand instructions for recovery
6. Reversal of actions – actions must be reversible
7. User In control – let the user 'feel' / be in control of the system at all times
8. Reduce short term memory – keep complexity low (ensure humans don't need to remember too much)

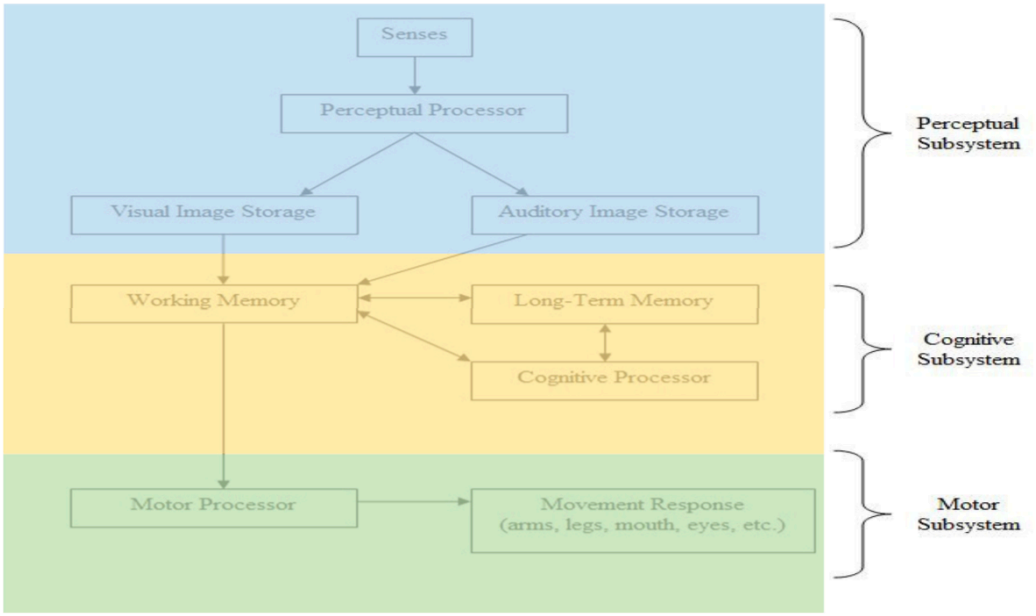
Cognitive Aspects for Design

Aim is to understand the concept and worldview of the user in order to design an interface for the system. We can user a model to understand how the user thinks and therefore make predictions.

Model Human Processor

Idea is there are three subsystems:

1. Perceptual
2. Cognitive
3. Motor



Perceptual Subsystem: Information is received, and responses are given via a number of channels

- Input Channels
  - Visual
  - Auditory
  - Haptic
- Output Channels
  - Movement
- Also, information is stored in memory
  - Sensory
  - Working (short-term) memory
  - Long-term memory

Cognitive Subsystem: Information is processed and meaning applied
Movement Subsystem: Motor processor leads to movement response

Processor Cycle Time:

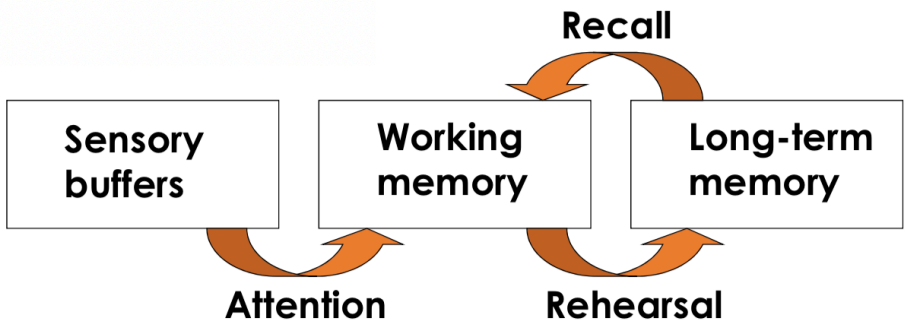
- Each processor has a cycle time
  - Tp = 100ms
  - Tc = 70ms
  - Tm = 70ms
- This allows a system designer to predict the time taken by a person to complete a task – determining the time of each operation

Memory

The Core Cognitive Aspects are:

1. Perception and Recognition
2. Memory
3. Reading, speaking and listening
4. Problem-solving, planning, reasoning and decision-making, learning

Three types of memory:



1. Sensory buffers
    a. Buffers from stimuli received through the senses – constantly overwritten by new information
    b. Information is quickly passed into more permanent memory store (working memory)
        i. By attention – concentration of the mind on one out of a number of competing stimuli or thoughts
    c. **Design Implications:**
        i. Cannot assume that because someone has seen or heard a particular message 5 seconds earlier, the person will remember it.
        ii. Keep message displayed until no longer needed
        iii. Then moves into working memory – where it will persist as long as 10 seconds
2. Short-term memory
    a. Working memory can be accessed rapidly – 70ms
    b. Transient
    c. Limited capacity – 7 ± 2 chunks of information
        i. STM constrained by number of chunks, not basic elements – therefore patterns can be useful as aids to memory
    d. Items lost from memory if not rehearsed
3. Long-term memory
    a. People store meaning or knowledge in long-term memory
    b. When people are recalling units of meaningful information they are placing items from LTM in an active state (working memory)
    c. Implications for Design
        i. **Context** plays a major role in what people see and hear
        ii. **Mind Set:** Factors that we know and bring to a situation can have profound effect on usability of an interface
        iii. **Grouping:** users will always automatically try to impose a structure on the display

Storage of Information:

1. Rehearsal
    a. Information moves from STM to LTM
2. Total time hypothesis
    a. Amount retained proportional to rehearsal time
3. Distribution of practise effect

    a. Optimized by spreading learning over time
4. Structure, meaning and familiarity
    a. Makes information easier to remember

Forgetting:

1. Decay
    a. Information is lost gradually but very slowly
2. Interference
    a. New information replaces old – they may interfere
3. Memory is selective
    a. Affected by emotion – can subconsciously choose to forget

Retrieval:

1. Recall
    a. When you remember something
2. Recognition
    a. When presented with the item you are trying to remember

Gestalt Psychology

Gestalt Theory: Describes how the mind organises visual data – do not see things in isolation but as parts of a whole

Gestalt Laws:

1. Figure-ground relationship – group elements as either figures or ground – this affects legibility
2. Proximity – group by distance or location
3. Similarity – group by type
4. Symmetry – group by meaning
5. Continuity – group by alignment
6. Closure – perceive shapes that are not there

Visual Perception

- Visual Perception is active – it blends sensation and knowledge – it is the active process of interpreting the information brought to the brain by the senses.
- We have lots of selective attention and perceptual expectancy

Reading

Several stages in the reading process:

1. Visual pattern of word is perceived
2. Then decoded with reference to an internal representation of language
3. Final stages of language processing include syntactic and semantic analysis and operate on phrases or sentences.

Expectation: What we expect to see affects what we perceive reality to be.

Colour: Perception of colour is determined by the physical context of the object

Colour Schemes:

- Different colour schemes are different combinations of colours based on relationship to each other
    - Monochromatic
        - Different tones of the same colour
    - Analogous
        - Based on colours that are adjacent to each other on colour wheel.
    - Complementary
        - Based on colours that complementary to each other on the colour wheel
    - Triadic
        - Based on three colours equally spaced around the colour wheel
- Any design should have a colour scheme

Rule Based Evaluation

GOMS and KLM

GOMS is the idea of describing the users behaviour in terms of:

- Goals
- Operators – perceptual, motor and cognitive acts
- Methods – procedure for using operators to accomplish goals
- Selection rules – if several methods available for single goal

GOMS Analysis checks that frequent goals can be achieved quickly (help us to compare different UI designs) – by making operator hierarchy. We also need to consider operator sequence – what order are operations done?

How?

1. Pick high level user goal
2. Write methods for reaching goals - subgoals
3. Write methods for subgoals, etc
    a. Until we reach operators

Operators have specific execution time, therefore summing the execution times mean we get time to perform serial operators. There are a specific set of operators for keystroke level model (KLM):

1. Keying – K
    a. 0.2s
2. Pointing – P
    a. 1.1s
3. Homing – H – time taken for a user to move hand from keyboard to mouse, or vice versa
    a. 0.4s
4. Mentally Preparing – M
    a. 1.35s
    b. Should be placed in front of all Ks and in front of all Ps that selects commands
    c. If operator following an M is fully anticipated in the operator just previous to that M, then delete the M.
    d. If a string of MKs belong to a cognitive unit then delete all Ms but the first

   e. No M between chars of a delimiter and character
   f. No M before 'enter' at the end
   g. Don't count any portion of an M that overlaps an R.
5. Responding – R – time user must wait for a computer to respond to input
   a. Any changes > 0.25s should have a response

+:

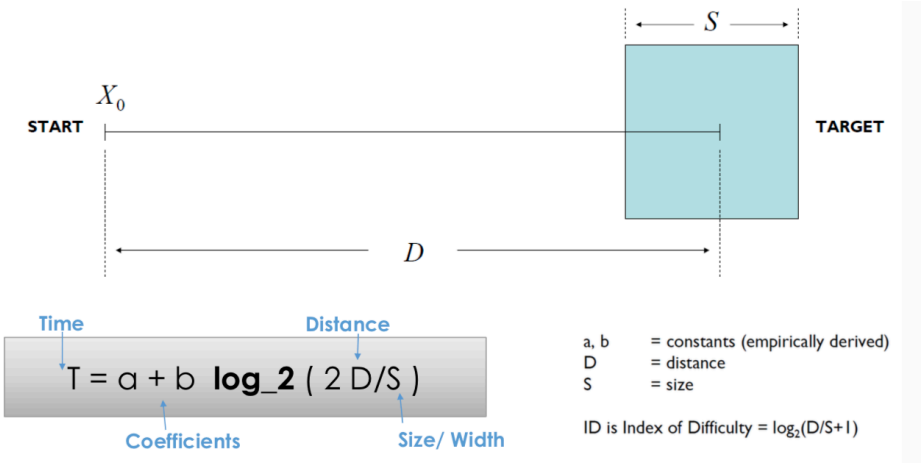
- Gives qualitative and quantitative measures
- Model explains the results
- Less work than user study
- Easy to modify when UI revised

-:

- Not as easy as other evaluation methods
- Takes lots of time and effort
- Only works for goal-directed tasks
- Assumes tasks expert performance without error
- Does not address several UI issues

Fitts' Law

Estimate movement time to select a target on a computer display – robust model of human psychomotor behaviour developed in 1954. It enables the prediction of human movement and human motion based on rapid, aimed movement. Movement time is affected by the distance moved and the precision demanded by the size of the target – precision measured by the index of difficulty.



The model considers a well-rehearsed selection task – T increases as the target increases and T decreases as the size of the target increases. A and b are determined experimentally and are mostly dependent on the pointing device.

Index of Difficulty

Index of Difficulty : **ID = $\log_2$ (D / S+1)** bit

D = distance or amplitude to move (length of a straight line from the position at which the cursor started to the closest point on the target)

S = size or tolerance of the region within which the move terminates

Applicability: Only applies to motions that are small relative to human body size and uninterrupted – one continuous motion.

Hicks Law

Method to estimate time taken to make a selection decision – more choices -> longer time taken to come to a decision:

$$DT = a + b \log_2 (n+1)$$

n → number of choices

A and b are dependent on many conditions, including how the choices are presented and how used to the system the user is:

- If choices are presented in a confusing manner, both a and b increase.
- Habituation decreases b

Heuristic Evaluation

Helps us to find usability problems in a UI – sketches or a real thing. We utilise a small set of evaluators (they do not need to be real people), each of whom checks for compliance with usability principles – heuristics. At the end, the problems are compiled and used to inform the re-design.

1. Visibility of System Status
    a. Keep users informed as to what is going on within a reasonable time
2. Match between system and real world
    a. Speak the users' language
    b. Ensure the terminology of your model meets their model
3. User control and freedom
    a. Idea that the user model is not necessarily shared by the system model
    b. No modal dialogs – system-imposed view – the user should not be forced down fixed paths
    c. Therefore, clearly marked exits - undo
4. Consistency and standards
    a. Consistency within (and between) applications
5. Error prevention

6. Recognition rather than recall
    a. Minimise the user's memory load – things should be easily visible and retrievable
7. Flexibility and efficiency of use
    a. Support frequent tasks with high cognitive load
    b. Allow people to do the same job in multiple different ways depending on how people think
    c. Macros, keyboard shortcuts
8. Aesthetic and minimalist design
    a. Draw users focus to the main subject – keep information displayed on the application simple
    b. Categorizing
9. Help users recognise and recover from errors
    a. Error messages in plain text
    b. Precise solutions
10. Help and documentation
    a. Modal, contextual help section (list of concrete steps)

Severity Ratings: Rating of how bad a problem something is in terms of: (1) frequency, (2) persistence, (3) impact:

- 0 – do not agree that it is a usability problem
- 1 – cosmetic problem
- 2 – minor usability problem
- 3 – major usability problem
- 4 – usability catastrophe

Performing Heuristic Evaluation

1. Pre-evaluation training – get evaluators up to speed on domain and scenarios used
2. Evaluate – evaluators individually use UI according to scenarios (twice: once for overview, once for detail)
3. Collate results
4. Rate severity
5. Feedback into design

+:

- Cheap and quick
- Easy to learn
- Finds lots of problems

-:

- Not task focused
- Not using actual people
- Not rigorous

Cognitive Walkthrough

CW is a task-centred evaluation, focussing on real, complete and representative tasks. The focus is on issues that users will have when they first use an interface, without training, therefore assesses **learnability.**

Why?

1. Question assumptions about what user thinking
2. Identify **missing / hard to find controls**
3. Finding places with **inadequate feedback**
4. Finding issues with labels or prompts
5. Comparison to other techniques
    a. Severe Problems
        i. Similar
    b. Content-related problems
        i. Comparable for consistency
        ii. Worse for recurrence
    c. Scope
        i. Finds more specific problems

Dependencies:

1. Description / prototype of the interface
2. Task description for a representative task
3. Complete list of actions needed to complete the task
4. Idea of who the users will be and their experience

Methodology

1. Define inputs
    a. Who are users
    b. What are tasks
    c. What are action sequences
2. Get analysts – do not need to be actual users
    a. Can also emulate a class of users
    b. Can be generally done by a developer
3. Step through action sequences for each task
    a. Will users know what to do?
    b. Will users see how?
    c. Will users understand whether their actions are correct or not (is feedback good?)
4. Record important information
    a. User knowledge before and after
    b. Side issues and design changes
    c. **Credible Success or Failure Story**
        i. **Did they / would they have managed it?**
5. Revise UI

-:

- Cannot evaluate every task the user will perform

- Each task is evaluated separately – no cross-talk interactions are identified
- Therefore, task-free, user-centred method is required to be brought in to catch problems that CW may have missed.
    - For example, heuristic evaluation