COMP27112 - Computer Graphics and Image Processing

November 26, 2017

1 Introduction

Computer graphics finds application in engineering, visualization, simulation and interactive virtual environments, such as AR.

Image processing finds application in engineering, medicine, face recognition media.

Lines have two kinds of properties: **geometry** (shape) and **attributes** (appearance).

A polygon is **convex** if it is not possible to find two points inside the polygon such that the line between them goes outside the polygon. Otherwise, it is said to be **concave** (non-convex). Since a concave polygon cannot be rendered correctly, we use **tessellation** to get a set of convex polygons from it, each of which can be correctly rendered.

2 Image synthesis

A **raster graphics image** is a dot matrix data structure, representing a generally rectangular grid of pixels. It uses a 2D array of pixels, in the case of screens, or dots, in the case of printers. Images must be sampled: the more samples, the better fidelity, even though it will always be an approximation.

A **vector display** is the first type of graphical display, with a Cathode Ray Tube and a controller. It draws vectors, and it is usually monochrome.

2.1 Double buffering

The idea here is that one buffer, called the **back buffer**, is only ever written to by the renderer. The other buffer, the front buffer, is only ever read by the DAC (Digital to Analog Converter). The renderer writes its new frame into the back buffer, and when that iss done, it then requests that the back and front buffers be swapped over. The trick is to perform the swapping while the DAC is performing its vertical retrace, which is when it is finished a complete sweep of its buffer, and is resetting to begin again. There is enough slack time here to swap the contents of the two buffers over. This method will ensure that the DAC only ever reads and displays a complete frame.

The swap between the buffers takes place during the **vertical retrace time** of the monitor, which is the time between the end of drawing of one frame and the beginning of drawing the next one.

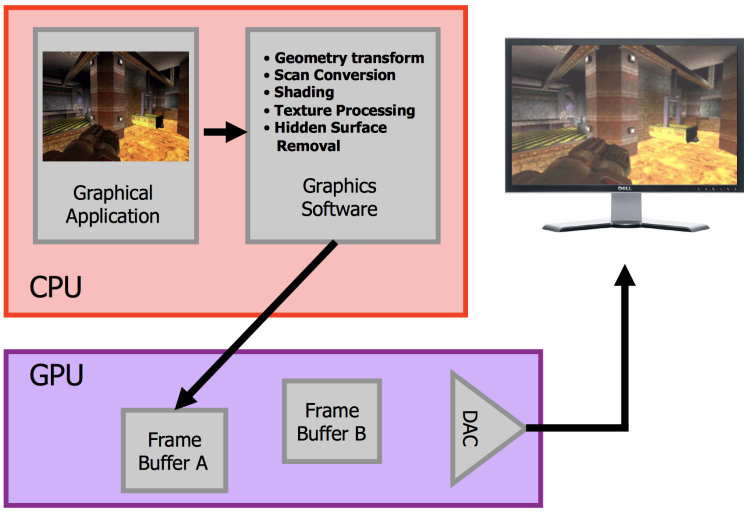

Figure 1: Example of basic graphics system architecture with double buffering

3 Transformations

A **coordinate** represents a point in space, measured with respect to an **origin** and the set of x, y, z **axes**.

A **vector** represents a **direction** in space, with respect to the 3D coordinate system. It has a characteristic length. A **unit vector** is a vector of length 1.

Both coordinates and vectors can be represented by a triple of x, y, z values. We define the shape of a geometric primitive using **points**. We can apply geometrical transformation to change them. To transform an entire shape, we transform all its individual points.

Transformations are expressed as 4×4 matrices. They can be combined and nested, allowing complex objects to be assembled from simpler parts.A transformation matrix used with column vectors is the **transpose** of the equivalent matrix used with row vectors.

In order to use a consistent matrix representation for all kinds of linear transformations, we need to add an extra coordinate w, which is in a 4^{th} dimensional space. Usually $w = 1$, but if not, it needs to be normalised. An **homogeneous coordinate system** has x, y, z, w axes.

Translation

Applies **3D shift** to coordinates

$$\begin{aligned} x_1 &= x + t_x \\ y_1 &= y + t_y \\ z_1 &= z + t_z \end{aligned}$$

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Scaling

Applies a **scaling factor**

$$\begin{aligned} x_1 &= x * s_x \\ y_1 &= y * s_y \\ z_1 &= z * s_z \end{aligned}$$

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation x-axis

$$x_1 = x$$
$$y_1 = y * \cos\theta - z * \sin\theta$$
$$z_1 = y * \sin\theta + z * \cos\theta$$

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

Rotation y-axis

$$x_1 = x * \cos\theta + z * \sin\theta$$
$$y_1 = y$$
$$z_1 = -x * \sin\theta + z * \cos\theta$$

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation z-axis

$$x_1 = x * \cos\theta - y * \sin\theta$$
$$y_1 = x * \sin\theta + y * \cos\theta$$
$$z_1 = z$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Algorithm 1 Rotation about an arbitrary 3D axis

Construct M_1 which translates A so that it passes through the origin $\rightarrow A_1$
Construct M_2 which rotates A_1 about the x-axis, mapping into the XY plane $\rightarrow A_2$
Construct M_3 which rotate A_2 about the z-axis, mapping it onto the x-axis $\rightarrow A_3$
Construct M_4 which applies the required rotation by θ about the x-axis
Construct the inverse matrices to undo the effects
return $P_1 = M_1^{-1} * M_2^{-1} * M_3^{-1} * M_4 * M_3 * M_2 * M_1 * P$

If we want to compose transformations (a **composite** transform), we multiply the transformation matrices. Matrix multiplications, and therefore transformations, are in general **non-commutative**, i.e. the order in which they are applied matters for the final result. One of the most common problems in computer graphics, blank screen syndrome, is often due to incorrectly ordered matrix transformations. Your image has been lovingly computed, but it is being displayed several miles to the West of your display screen; or that tiny blob in the left-hand corner of your screen is your image, compressed into a few pixels.

Two matrices A and B are **inverses** of each other if $A \times B$ is an **identity matrix** I. *Not all matrices have an inverse.* A transformation is **singular** if it throws away information i.e. it does not have an inverse and it cannot be undone.

An object is defined in a **local modelling coordinate system**. **Modelling transformations** are used to **instance** multiple copies of an object in the scene. A **global world coordinate system** is used to specify the position of objects in the entire scene.

Normalisation is the process of taking an arbitrary non-zero vector and converting it into a vector of length 1. You calculate the initial length and divide x, y and z by it.

The **dot product** is the scalar product of individual components. For normalised vector, it is the cosine of the angle in-between.

The **cross product** of two vectors is a third vector perpendicular to them both.

4 Polygons and pixels

The basic unit of 2D raster graphics is the **pixel**. The most common basic unit of 3D graphics is the polygon, the triangle in particular. A **polygon** is the space bounded by a set of edges between each pair of vertices in an ordered set. It has a **front** and a **back**. It has an attribute called **winding**, which is the order in which the edges connect the vertices. It can be **clockwise** or **counter clockwise**. Winding must be *consistent* for all the polygons in a scene. A **planar** polygon has all the vertices lying in one plane. Triangle are always planar.

The **surface normal** is the vector perpendicular to the plane of the polygon. It is used to give a polygon a front and a back, and it is used to describe its orientation in 3D space.

Algorithm 2 Finding the surface normal

Choose a pair of sequential edges and compute their vectors
Invert the direction of the first → now they emanate from the shared vertex
Their cross product gives the surface normal
Normalise the surface normal

A **polygon soup** is a huge list of unorganized individual polygons, coloured and drawn in order, with no relation whatsoever, requiring individual manipulation. It is a waste of storage space, as polygons could share their vertices. There is also a complete loss of semantics. It leads to brute force rendering, and makes interaction with the model complex.

A **mesh** is a (often big) linked group of polygons to represent surfaces. It retains semantics, helps interaction with the model and reduces storage. A **face list** can be indexed into the **edge list**, which can be indexed into the **vertex list**.

- **Triangle strips**: collections of linked triangles. N triangles are defined using $N + 2$ edges.
- **Triangle fan**: collections of linked triangles sharing the same vertex.
- **Quadrilateral strips**: collection of linked quadrilaterals. N quadrilaterals can be defined using $2N + 2$ vertices. $N \times N$ quadrilaterals may be defined by $(N + 1) \times (M + 1)$ vertices. Quadrilaterals must be tessellated into triangles during rendering.

Scan conversion is the process of converting a polygon edge described by world coordinates to an edge on the screen described by pixel coordinates.

To **scan-convert a line**, we sample the true geometry and approximate it using the nearest pixels available. Bresenham (1965) developed a fast algorithm using only integer arithmetic, still in use today.
To **scan-convert a polygon**, the polygon is transformed by the viewing pipeline, so we know its (x, y, z) vertex coordinates in screen space. The (x, y) coordinates correspond to a pixel position. The z coordinate is a measure of the vertex's distance from the eye.
There are two approaches to **scan-convert a triangle**. The first one consists in scan-converting each of the edges, then processing each row of pixels and fill in the remaining interior pixels. The second approach instead is called **sweep-line algorithm**. It steps down a pair of edges, goes down scanline by scanline inside the triangle and then fills the pixels inside the triangle for that scanline. This algorithm is efficient, as we only compute the slopes of the edges once (at the beginning). However, it is a floating point algorithm, so we have to keep rounding to the pixel grid.

When modelling a 3D world, we have to take into account that some surfaces are non-visible as they are behind other surfaces. We can solve this problem in display space: during scan conversion, whenever we generate a pixel P, we determine whether some other vertex, closer to the camera, also maps to P. the closest one will be drawn, the others won't.

The **z-buffer**, also called **depth-buffer**, keeps a record of the z-value of each pixel. Lack of precision in the z-buffer leads to incorrect rendering of pixels with similar z-values: **z-fighting** and **stitching**.

Structured polygons maintain a hierarchy of structure: object → surface → polygons → edges → vertices.

Algorithm 3 The z-buffer algorithm

```
Initialise each pixel to desired background colour
Initialise each z-buffer entry to MAX_DEPTH, which is the biggest possible number
for each pixel P generated during scan-conversion of an object do
    if (z-coordinate of P < z-buffer[P]) then
        Compute colour of P
        Store colour of P
        Store z-coordinate of P in z-buffer
    else
        Do not change P    // Something else has already mapped to P and is nearer to us
```

5 Viewing

5.1 The camera

To display a scene on the screen, we use the analogy of photographing the scene with a camera, simulated by transforming the model. We specify a **window** in world coordinates, and a **viewport** in screen coordinates: $P_{screen} = M_{view} \times P_{world}$. M_{view} transforms the window to the viewport and it is called **viewing transformation** ($M_3 \times M_2 \times M_1$).

M_1 places the window at the origin with a translation. M_2 scales the window to be the same shape as the viewport. M_3 shifts to viewport position.

We want to **clip** against the viewport to remove those parts of primitives whose coordinates are outside the window. Sometimes it is useful to use multiple windows and viewpoints, to help arrange items on the screen. In order to view a 3D world on the screen, we have to reduce 3D information to 2D information.

Algorithm 4 The camera analogy

```
Arrange the scene into the desired competition → Set Modelling Transformation
Position and point the camera at the scene → Set Viewing Transformation
Choose a camera lens → Set Projection Transformation
Decide the size of the final photograph → Set Viewport Transformation
```

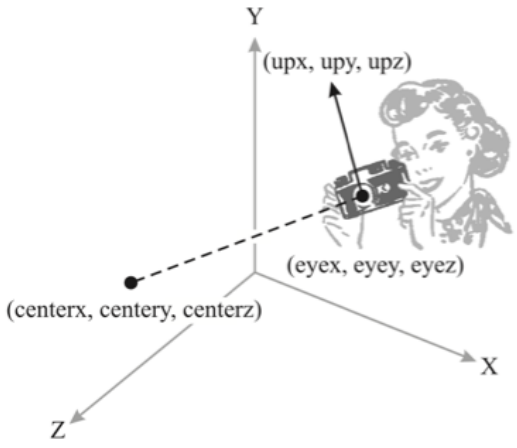

Figure 2: The OpenGL camera

When we say "changing camera and orientation", we actually compute a viewing transformation which we then apply to the object

The duality of Modelling and Viewing: we can create the same view from a camera at a certain location and orientation by transforming the object.
Moving the model by (x,y,z) is equivalent to moving the camera by $(-x,-y,-z)$.
The camera is specified by:

- **Eye point** (E): where the camera is located in 3D space.
- **Centre of interest** (C): a 3D lookpoint at which the camera is looking.
- **Up** (V): direction of the camera, using a view up vector.

There is no need for the view vector and the up vector to be defined at right angles to each other (although if they are parallel, weird views may result). Often the up vector is set to a fixed direction in the scene, e.g. pointing up the world Y axis. In the general case, OpenGL twists the camera around the view vector axis until the top of the camera matches the specified up direction as closely as possible. We use the eye point, the centre of interested and the view up vector to derive a transformation which, when applied to the model, would give the same view as if we had a real camera.

We also use them to derive a coordinate system $(s,\ v,\ f)$ for the camera:

- **F**: $C - E$ - normalised vector since it specifies a direction only.
- **S**: normalise $(F \times U)$.
- **V**: normalise $(F \times S)$. We do not make any assumptions that the user made sure V is orthogonal to F.

5.2 Projections

Projection is the process of converting all 3D coordinates in suitable 2D ones. There are two classes of planar geometric projection: parallel projection and perspective projection.

5.2.1 Parallel projection

In **parallel projection**, the projection is a set of points at which the projector (aka the object's vector) intersects the projector plane. Parallel edges in the original object remain parallel in the projection, while angles between edges may be distorted. The centre of projection (the "eye point") is at infinity, so the projectors are parallel.

In **orthographic** parallel projection, projectors are perpendicular to the projection plane, which is parallel to one of the axes of the 3D-object. The projected image shows only two axis, and no distortion of lengths or angles occurs. There is a scaling by 1 for the visible axes. It is suited for engineering drawing and CAD.

In **axonometric** parallel projection, projectors are perpendicular to the projection plane, which can have any orientation relative to the object being viewed. All of the three axes can be seen. Distortion of lengths and angles may occur. It is **isometric** if the projection plane is symmetrical to 3 of the (x,y,z) axes. It is **dimetric** if the projection plane is symmetrical to 2 of the (x,y,z) axes. It is **trimetric** if the projection plane is symmetrical to 1 or 0 of the (x,y,z) axes.

In **oblique** parallel projection, projectors can make any angle with the projection plane and the projection plane can have any orientation relative to the object viewed. It is the most general case of parallel projection. We see all of the three axes. Distortion of lengths or angles occurs.

We can derive matrices to perform all parallel projections.

5.2.2 Perspective projection

A **perspective** projection reflects the way we see it, giving a sense of realism. A projection is a set of points at which the projectors intersect the project plane. Object further away from the centre of perspective become smaller. Edges that were parallel may converge. Angles between edges may be distorted. The number of (x, y, z) axes parallel to the projection plane determines the number of **vanishing points** are seen in the projected image: **1-point perspective** with 2 parallel axes, **2-point perspective** with 1 parallel axis and **3-point perspective** with 0 parallel ones.

We may derive the matrix to perform a 1-point projection where the projection plane is parallel to the XY plane. We have renamed d to d_z just to remind us that it is a z-coordinate.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d_z & 0 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

We can also derive a matrix which expresses the most general case, 3-point projection, where the projection plane is not parallel to any of the XY, XZ or YZ planes. The projection plane intersects the (x, y, z) axes at (d_x, d_y, d_z).

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/d_x & 1/d_y & 1/d_z & 0 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

As we need $w = 1$ to end up with a 3D point, we need to divide all the elements by w_1. This is called **perspective division**.

Consider the real world camera analogy, in which we choose the lens type. The choice of lens affects the **field of view**, and selects what portion of the 3D world will appear within the bounds of final image. The volume of space which eventually appears in the image is known as the **view volume**. As well as discarding objects which lie outside the image frame, OpenGL also imposes limits on how far away objects must be from the camera in order to appear in the final picture. The actual 3D shape of the view volume depends on what kind of projection is used. For orthographic (parallel) projections the view volume is box-shaped, whereas perspective projections have a view volume shaped like a truncated pyramid. The facets encasing the view volume effectively define six clipping planes, which partition the frustum interior from the unseen outside world.

We would like to have a perspective transformation which preserves depth. Suppose we have a particular perspective transformation expressed by frustum F and matrix M. It can be shown that we can derive a new transformation matrix PN, based on M, that distorts F into a cube. Transforming our model by PN and then taking an orthographic projection produces exactly the same result as performing our original perspective transformation, M, but this time preserving the z-depth value. This is called **projection normalisation**. **Clipping** takes place in the cube produces by projection normalisation.

5.3 3D viewing

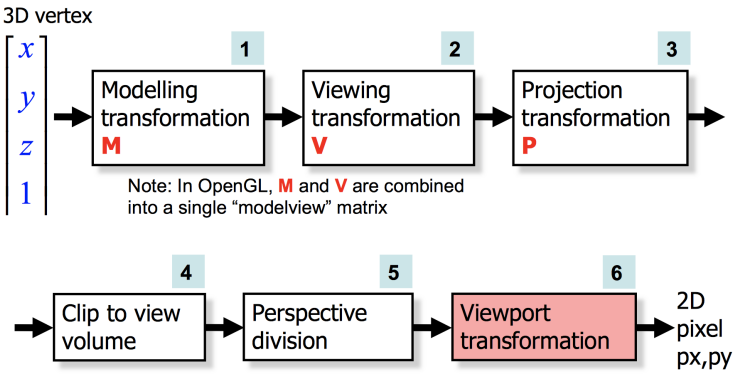

Figure 3: The 3D viewing pipeline

Algorithm 5 3D viewing

The Modelling Transformation arranges objects in our 3D world
The Viewing Transformation transforms the world to give the same view as if it was being photographed by a camera
The Projection Transformation performs a parallel/perspective projection within **clip planes**
Parts outside clip planes are discarded
if it is a perspective view **then**
    the perspective division "flattens" the image
The Viewport Transformation maps the final image to a position in part of the display screen window

6 Rendering

We can make models of light-matter interaction in two ways: in a **local illumination model**, we treat each object in the scene separately form any other object, the reflections between objects are ignored; in a **global illumination model**, we treat all objects together and modelling the interactions between them. In computer graphics, we try to model the interaction of light an matter approximating it (more realistic yet more complex). Computer pictures are digital, with finite precision, thus we can only ever approximate.

6.1 Reflectivity

There are three kinds of reflection: perfect diffuse reflection, perfect specular reflection and imperfect specular reflection.

Diffuse reflection is absorption and uniform re-radiation. Some wavelengths are absorbed while others are reflected. Diffuse reflectors always absorb some specific wavelengths and reflect others, which is why they have an unchanging colour.
Specular reflection is reflection at the air/surface interface. The colour of the specular reflection is the one of the light source.
In **perfect diffuse** reflection, incident rays of light are reflected in all directions from the surface. A perfect diffuse surface reflects an incoming ray across all angles. The surface looks dull, or matte. Example of it are a carpet, a sponge, a brick.

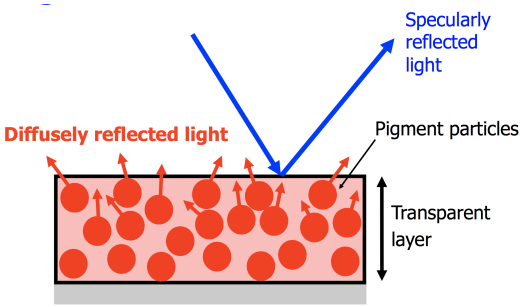

Figure 4: Difference between diffuse and specular reflection

In **perfect specular** reflection, a surface reflects an incoming ray of light like a perfect mirror. The surface must be smooth.
In **imperfect specular** reflection, a surface reflects an incoming ray across a small range of angles. The surface is irregular, shiny, with highlights. Examples of imperfect specular reflectors are stainless steel, glazed ceramic and lacquer-coated aluminium.
Most surfaces exhibit a combination of the two.

6.2 Illumination

Consider an environment with a light source. Multiple reflections cause a general level of illumination in the scene, called **ambient illumination**. It has the effect that each object is uniformly illuminated, but 3D information are lost. The amount of light diffusely reflected from a surface is

$$I = k_a \times I_a$$

where k_a is the **ambient reflection coefficient** of the surface, $0 \leq k_a \leq 1$, and I_a is the intensity of ambient light. Ambient illumination is not constant in the scene, so this is a gross simplification.

If we consider a point illumination source at infinity, the direction is our only concern, and this is called **directional lighting**. We need to model the effects of different angles of incidence and different distances from the light source.

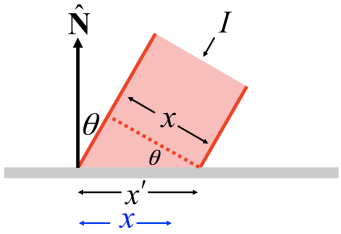

Figure 5: Lambert's Law derived

Consider light of intensity I and cross-sectional width x falling on a surface, so width x on surface receives all of intensity I. Now consider the light inclined at θ. $x' = x/\cos\theta$, thus $x = x'\cos\theta$, and so width x receives intensity $I\cos\theta$. This is called **Lambert's Law**. Thus, the updated local illumination model which takes into account both ambient and diffuse illumination is

$$I = (k_a \times I_a) + (I_p \times k_d \times (N \times L))$$

where I_p is the intensity of the light source, k_d is the **diffuse reflection coefficient** of the surface, $0 \leq k_d \leq 1$, N is the surface normal and L is the direction of light (both vectors are normalised).

Physically, light intensity falls off with the square of distance travelled. After travelling d, the original intensity I_p becomes I_e. While this is physically correct, it does not always work well for computer graphics. We only have a limited number of pixel intensities, and often the d^2 term changes too rapidly, so instead we use $I_e = \frac{I_p}{k_c + k_l d + k_q d^2}$ (inverse square law). We can choose k_c, k_l and k_q for the best results. Thus, the updated local illumination model which takes into account both ambient and diffuse times the distance illumination is

$$I = (k_a \times I_a) + \left(\frac{I_p}{k_c + k_l d + k_q d^2} \times k_d \times (N \times L) \right)$$

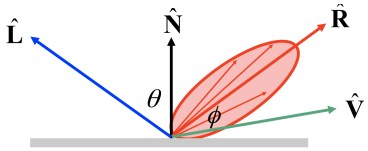

Figure 6: Modelling specular reflection

where R is a vector giving the direction of maximum specular reflection, which makes an angle of θ with N, as L does. V is a vector pointing to the observer's position. As V diverges from R by angle ϕ, viewer sees less specular reflection.

The updated **local illumination model** which takes into account is

$$I = (k_a \times I_a) + \left(\frac{I_p}{k_c + k_l d + k_q d^2} \right) \times k_d \times [(N \times L) \times (k_s * (R * V)^n)]$$

where k_s is the **specular reflection coefficient** of the surface, $0 \leq k_s \leq 1$, and N is the strength of the specular reflection. It must be noted that $(R * V)^n = \cos^n \theta$, is the **Phong term** and θ is the angle between light ray and the specular reflecting surface.

This is then

$$ambient + diffuse \times (distance + specular)$$

If we have more than one light, we compute the illumination separately for each and perform a sum.

6.3 Shading

6.3.1 Flat shading

Also known as **constant shading**, it is known as the simplest approach. It consists in computing colour C at one vertex and use it for all pixels in the polygon. Each polygon is uniformly coloured according to its orientation and we clearly see the mesh.

The **Mach banding** effect consists in the edges between strips of different intensities to stand out. It is an optical illusion that exaggerates the contrast between edges of slightly different shades of grey, as soon as they contact one another, by triggering edge detection in the human visual system.

6.3.2 Gourand shading

Also known as **intensity shading**, it has been invented by Henri Gourand in 1971. It uses intensity interpolation to smooth out the discontinuities between polygons. We can approximate the normal of the underlying surface by averaging the normals where polygons share vertices.

You also compute pixel C_a, C_b and C_c colours of vertices A, B and C, and for each scanline you average C_a and C_c in C_{left}, you average C_b and C_c in C_{right} and finally average C_{left} and C_{right} along the scanline. Mesh now appears smoother, but the silhouette edges are still polygonal. Even though this method is fast and efficient, specular highlights may be distorted or averaged away altogether since Gourand shading averages between vertex colours. Mach banding may still be visible. Sometimes edges are shaded away, thus edges must be tagged in the data structure to avoid interpolationa across it.

6.3.3 Phong interpolation

Also known as **normal-vector shading**, consists in interpolating the normal vector along the scanline and computing the illumination model for every pixel. For each scanline, we compute the average normal N_{left} from N_a and N_c, we compute the average normal N_{right} from N_b and N_c, we average between N_{left} and N_{right} and compute the colour of the pixel using the illumination model. Specular highlights are rendered correctly, but rendering is more expensive.

Figure 7: Flat shading vs Gourand shading vs Phong interpolation

6.3.4 Rendering expenses

Our local illumination model takes about 60 floating-point operations to compute a colour for pixel. For a Gourand-shaded triangle, that's 180 floating-point operations, about 2 per pixel. For a Phong-shaded triangle, that's 60 floating-point operations per pixel.

6.4 Texture and surface details

A **texture** is defined as a 2D array of **texels**, often read from an image file or hardcoded for regular patterns. it is defined in its own coordinate system, conventionally referred to as (u, v). We associate (u, v) texture coordinates with each (x, y, z) vertex of a polygon and interpolate the texture coordinates during scan-conversion. Then, we blend the pixel colour with the texture

colour. We perform texture mapping as part of **rasterisation**, sampling the texture at each pixel. In order to realistically texture a mesh, we often have to use multiple textures in different places which can give rise to ugly seams. One solution is to use textures that are **seamless**, so the edge of one exactly matches the edge of another.

Textures can be used to add accurate illumination to a real-time scene offline, computing accurate diffuse illumination model of the scene, or in real-time, applying lightmap textures.

A **resolution mismatch** occurs when the texture resolution does not match the pixel resolution.

6.4.1 Resolution mismatch: pixel resolution > texture resolution

Texels are bigger than the pixel, thus pixels A and B happen to map the same texel.

No filter: we simply select the texel to which the pixel maps. The resulting image looks blocky.

Bilinear interpolation filter: we compute a texel colour from adjacent texels, averaging horizontally and vertically. The resulting pixel image looks smoother.

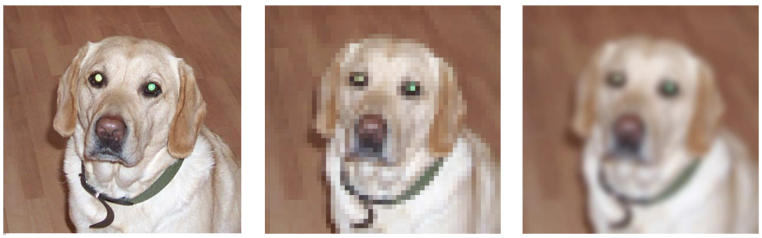

Figure 8: 1 to 1 correspondence vs. no filter vs. bilinear interpolation filter

6.4.2 Resolution mismatch: texture resolution > pixel resolution

Pixels are bigger than texels, thus adjacent pixels might map to texels further apart in the texture, leading to missing detail (**aliasing**). In animated sequences especially, we see unpleasant aliasing effects, as pixels pop on and off, or change colour unexpectedly in each frame.

Mipmap filtering: mip comes from Latin, "multum in parvo", which means "many things in a small place". The further away the object is from the viewpoint, the less detail is needed. So we use a set of texture maps, and select which map to use, according to the distance of a pixel from the viewer. Starting from the original texture, we repeatedly create smaller versions of it, downsampling by $1/2$ each time, until we reach a 1×1 texture.

This is a **pre-processing** procedure, and we store each texture in memory. When rendering, we select one of the textures according to the distance of the pixel from the viewpoint. Alternatively, we can also choose the two closest textures, and do bilinear interpolation, for extra smoothness. With mipmapping, the aliasing is replaced by smoothing.

6.4.3 Bump mapping

Bumpy surfaces look bumpy because the surface normals change across the bumps, so the top of the bumps appears brighter than the sides. Rather than alter the surface colour, as in texturing, we can alter the surface normal which has the same effect on the illumination model as if the surface were really geometrically different.

We can draw a texture to use as a **bump map** to derive b_u and b_v. We treat the value of each texel as the height which control the bumpiness of the surface. For each texel T, we can compute its x and y gradient and use them as b_u and b_v.

- b_u: $T(x-1, y) - T(x+1, y)$.
- b_v: $T(x, y-1) - T(x, y+1))$.

Algorithm 6 Altering a surface normal

Introduce two unit vectors N_u and N_v, each orthogonal to the surface normal
Scale N_u and N_v by b_u and b_v to give the bump vectors
Add bump vectors tot he original surface normal to get an altered one

7 Image representation

The human visual system provides a useful analogy with computer vision systems. The visual world is perceived because light reflected from surfaces in it is focused on to light sensitive cells in the eye. In dark conditions, more light is needed to enable us to see clearly and consequently the iris dilates. The iris contracts in bright conditions so admitting less light. This is analogous to the case of a photographic camera; exposure control is achieved by choosing film of varying sensitivity and by altering the aperture of the lens. Three factors influence the quality of the perceived data: the angular resolution of the eyes, the colour sensitivity and the brightness sensitivity.

7.1 Image capture

Anything that can generate a spatially coherent measurement of some property can be imaged. Images may be captured using a wide range of instruments, but they all share the property of being able to transform some type of energy into a visible form. If a video camera is used for image conversion, it will transform the illuminated scene into an electrical signal. An analogue video camera will output a standard PAL or NTSC signal; a digital camera will output digitised data. The analogue signal must then be converted into a digital format. Whatever type of detector is used to capture the image data, it will transform a continuous brightness function into a discretely sampled image. An image is formed by the capture of radiant energy that has been reflected from surfaces that are viewed. The amount of reflected energy is determined by two functions: the amount of light falling on a scene and the reflectivity of the various surfaces in the scene. The two functions are known as the illumination, $i(x, y)$, and reflectance, $r(x, y)$, components. The **illumination** function can take values between zero and the maximum ambient illumination. The **reflectivity** function can take values within the range zero (complete absorption of incident energy) to one (complete reflection of incident energy).

7.2 Sources of degradation

The preferred definition is "anything with the imaging system that causes a change", such as electrical interference and optical aberration.

- Noise in the electronics that converts radiant energy to an electrical signal.
- **Geometric distortion**: occurs because the best focus of the scene in front of the lens is in fact on spherical surface behind it.

- **Scattering**: occurs when rays of light reflected by the surface to be viewed are further dispersed by material in the optical path between the surface and the detector.
- **Blooming**: occurs when a bright point source is imaged, the image is blurred slightly due to the camera's optical system and neighbouring cells.
- Overflow or clipping of a signal occurs whenever the signal is of excessive amplitude.

Noise is any deviation of the signal from its expected value. It may be random ($x \to x + n$). **Pepper noise** ($x \to \{max, min\}$) randomly introduce pixels of pure black and pure white into the image. **Gaussian noise** has its amplitude distribution described by a Gaussian or normal distribution: it will usually have zero mean and finite amplitude. The **histogram** describes the frequency with which each grey level or each colour occurs in the image. The standard deviation of the histogram may be used as a measure of the noise amplitude. The **signal to noise ratio**, usually quoted in decibels, compares the noise amplitude against the amplitude of the noise free data.

$$SNR = 20 \log_{10} \frac{signal}{noise}$$

7.3 Image resolution

There are three resolutions: spatial (loosely the number of pixels), brightness (the number of shades of grey or colours in the image) and temporal (the number of frames captured or processed per second).

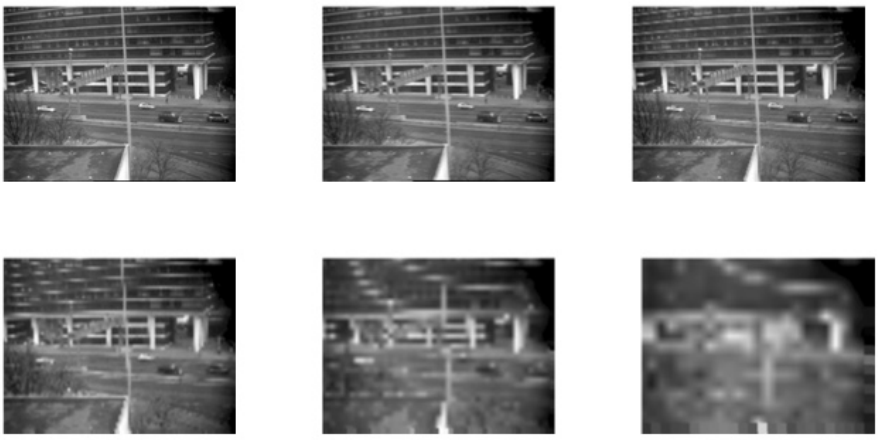

Figure 9: Spatial resolution at n, n/2, n/4, n/8, n/16, n/32 pixel

The definition of **spatial resolution** must consider the interaction between the optics of the camera and the detector. The **angular resolution** can be used to specify how close an object must be for it to be resolved (seen), or what objects can be resolved at a certain range. It is equal to the ratio of the smallest resolvable object to the range. An alternative way of specifying the spatial resolution uses a a variant of **Nyquist's theorem**. Nyquist's theorem states that a periodic signal can be constructed if the sampling interval is half the period and an object can be detected. Thus, the variant that can be used in ours case is: if two samples span its smallest dimension two pixels must span the smallest dimension of an object in order for it to be seen in the image. If a signal is sampled at the Nyquist rate, little information is gained, other than knowledge of the presence of the signal. Similarly, if an image is sampled at this rate, the presence of objects of this size is determined. To discover more about the objects requires higher sampling rates.

The **brightness resolution** of a system can be defined as the number of shades of grey that may be discerned in a monochrome image, or the number of distinct colours that may be observed in a colour image. The major contributor of noise in a digital system is the camera; the analogue to digital conversion adds very little noise. By viewing typical images using varying numbers of shades of grey or colour we may also estimate the number of grey values required in an application. False contours are visible in areas of gradually changing brightness, when there are insufficient shades of grey. The human visual system being unable to differentiate more than about 40 shades of grey.

Temporal resolution is defined either as the number of images that are captured and stored per second, or as the number of images captured and processed per second.

It is helpful to consider spatial and brightness resolution separately, to avoid confusing the effects of these causes. However, the two resolutions can interact in affecting the overall perception of an image: a poor spatial resolution can sometimes be compensated for by a good brightness resolution. In fact, for each brightness resolution it is usually possible to define a threshold spatial resolution above which the image appears of an acceptable quality and below which it does not.

7.4 Connectivity and distance

Four-connectivity: a pixel is connected to the four nearest neighbours (north, south, east, west). Objects joining at corners can be disconnected.

Eight-connectivity: a pixel is connected to the four nearest neighbours and the four next nearest (i.e. ne, se, sw, nw). It can pierce thin objects.

Euclidean distance: $D(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
City-block distance: $D(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$

7.5 Colour representation models

Newton suggested that there were seven colours of light that generated white when mixed.
However, Thomas **Young** in 1801 proposed that there were in fact three identifiably different colour receptors. The basis of Young's theory was the observation that most colours could be matched by mixing different proportions of a red, a green and a blue primary source. Negative amounts of red are required to match certain colours.

$$C = rR + gG + bB$$

RGB colour space is **perceptually non-linear**. If we change a colour's RGB value by a fixed amount, the perceived effect it generates is dependent on the original RGB values. It is also difficult to predict the change that will occur to a colour by changing one of its components by a certain amount. Equal distance on CIE does not correspond to equal changes in perceived colour.

Other colour representation models suggested include YCrCb, whichp takes into consideration intensity and colour difference and it is widely used in broadcasting, and perceptual spaces.

HSV is a perceptual colour space which uses three components:

- The underlying colour of the sample, the hue (H).
- The intensity of the sample or its brightness (V).
- The saturation or depth of the sample's colour (S).

Thus, reddish colours will have similar hue values but will be differentiated according to their saturation.

Lab can represent all perceivable colours. It is device independent. L represents lightness, a represents green → magenta and b represents blue → yellow.

IHS takes into consideration the intensity (aka brightness, weighted average of RGB), the hue (the basic colour, an angle between 0 and 359) and the saturation (aka the depth of colour).

All of them separate intensity/brightness and chromaticity.

7.6 Camera calibration

The **calibration** process divides into two stages, one that relates image co-ordinates to a co-ordinate frame aligned to the camera, and one that relates the camera co-ordinate frame to an arbitrary co- ordinate frame fixed to some point in the physical world.

Extrinsic parameters relate to the camera to world co-ordinate transform and **intrinsic parameters** relate to the image to camera co-ordinate transform.

8 Image transform and feature extraction

An **image processing operation** is defined simply as an operation that somehow manipulates image data to generate another image. Some classify algorithms according to their task, but this approach assume prior knowledge: if we are able to identify the task we have to perform, we can discover what algorithms are needed. It is also possible to present algorithms according to their place in the processing hierarchy. Thus the description would follow the logical order of the algorithm's use.

The aims of the image transformation are as follows:

- To remove or correct for degradations introduced by the image capture process.
- To improve the appearance of the image for human perception or for further processing.
- To identify and quantify structures in the image that may be indicative of the objects in the scene being viewed.
- To transform the image into an alternative representation in which some operations may be performed more efficiently.

8.1 Point transforms

A **point transform**, or a **grey value transform**, will manipulate the grey or colour value of an individual pixel without considering the neighbouring values. Every input value must have an associated output, but this need not be unique: more than one input might share the same output.

The two simplest methods of manipulating the grey scale of the image are to add a constant to all values (**brightness adjustement**) and to scale all values by a constant multiplier (**contrast adjustement**). These methods are arbitrary: how much should be added to the image or what scaling factor ought to be used is determined specifically for each image being processed according to the desired outcome. To reduce this arbitrariness, the images are often manipulated such that the mean grey level is matched to a previously determined value. The grey scale of the image can also be manipulated interactively such that certain portions of it are expanded at the expense of others that are compressed: the extreme values of the scale, normally 0 and 255,

will always remain unchanged. The aim of manipulating the grey map in this way is to expand certain portions of its range and thus make the objects that have this range of brightnesses appear more clearly in the manipulated image. The problem with the techniques discussed is that they require an operator to recognise what range of brightness should be expanded.

False contouring applies a grey scale to an image. The aim of this transformation is to enhance gradual changes in brightness by changing them into a succession of rapid changes from dark values to bright values.

The **grey level histogram** records the frequency of occurrence of each grey value in the image. The cumulative histogram records the number of pixels in the image with grey values less or equal to each index, and it is derived from the grey level histogram.

Histogram equalisation is based on the argument that the image's appearance will be improved if the distribution of pixels over the available grey levels is even. Equalisation transformation warps the horizontal axis of the histogram, stretching it in some regions and compressing it in others, such that the frequencies of grey values become more uniform. However, it is not the best method of improving the image's appearance: the histogram should be hyperbolised.

8.2 Local transforms

Local transforms are those that are computed using a small region surrounding the pixel of interest.

8.2.1 Convolution

$$g(r,c) = \sum_{x=\infty}^{\infty} \sum_{y=\infty}^{\infty} f(r-x, c-y)t(x,y)$$

A usually small **template**, $t(x,y)$, is placed over all possible image locations. At each location, the product of an image value and the overlapping template value is computed. The products are summed to give the output value at that location. Optional further steps are to subtract an offset, to guarantee that the minimum output value is zero and to scale the result such that the maximum value will lie within the range allocated to an image value. The **offset** to be subtracted would be the smallest result that could be computed. The multiplicative scaling factor could be the ratio of the maximum image value to the largest result that could be expected or it could be the sum of the template values. The dimensions of the template define the practical limits of the summation. The template is defined analytically, as is its separation. Convolution of three templates can be performed in any order.
Convolving with a $n \times n$ template (i.e. Laplacian template) results in n^2 multiplications and additions per output pixel.
Convolving with a $n \times 1$ template (i.e. separated kernels) results in $2n$ multiplications and additions per output pixel, thus it results in faster processing.
Convolution is distributive. We can create a composite filter, having an efficiency gain.
It has four main applications: smoothing (i.e. noise reduction), sharpening (i.e. edge detection), corner detection and template matching (i.e. finding objects).

8.2.2 Smoothing

$$\sum(x+n) = \sum(x) + \sum(n) \approx \sum(x)$$

The effect of **smoothing** on an image is to remove **sharp**, sudden changes in the brightness function. These might be caused by noise in the image capture device or small objects in the scene that might obscure the larger objects of the scene. The result of smoothing an image is another image with an improved signal to noise ratio, or an image in which the effect of distracting artefacts has been reduced. The value of a smoothed pixel in an image is computed by combining a selection of the neighbouring pixels. The simplest method of achieving the combination is simply to average the values in the neighbourhood. The neighbourhood that is to be used defines the size and shape of the template. The template values are simply the reciprocal of the number of values included in the neighbourhood. Noise amplitude is reduces by template length. After smoothing, edges are no longer as sharp as they were, and this is an undesirable side effect. This effect might be reduced by conditional smoothing: the smoothed value is computed, but it will only replace the original if it differs by less than a preset value.

Local smoothing removes details and introduce ringing.

Adaptive smoothing: $output = \begin{cases} s & |s-x| < T \\ x & otherwise \end{cases}$

Rank-filters are used in preference, but at a higher computational cost. A rank filter will select one value as the smoothed value from the set of pixels in a neighbourhood that have been ranked according to their magnitudes. An example of it is median filtering.

Algorithm 7 Median filtering

Choose a window size for smoothing i.e. 3×3 pixels
Process the image so that you always find a 3×3 window of pixels. The edge in this case is 1
When you encounter a pixel, store its entire window in an array
Sort the array
Pick the element in the middle of the array, in this case at index 4, and assign its value to the current pixel
Repeat the first step for all the pixels in the image it can be applied to

Gaussian smoothing reduces ringing using weighted smoothing. Weights are derived from Gaussian (normal) distribution.

8.2.3 Sharpening

Also referred to as **edge enhancement**, it intends to exaggerate the effect of significant local variations in the image data. It thus enhances discontinuities and edge detection. It is both perceptually and computationally important. Edges define the significant structures in a scene. There are step edges, line edges and roof edges.
An **edge** is a significant, local, extended change in image intensity:

- **Significant**: the difference between two pixels in relation to their brightness. There will be a monotonic change from significant to insignificant as the proportional change in image intensity across the edge decreases i.e. not just two pixel values brighter.
- **Local**: the difference between two adjacent pixels will be of greater significance than the same difference between two pixels separated by the image's width
- **Extended**: the edge is not simply a local variation that could have arisen by random fluctuations of the data.

If an edge is a discontinuity, we can detect it by differencing and applying convolution with appropriate templates.

The **Robert cross edge-enhancement operator** is very noise sensitive: if one pixel is corrupted, the edge strength is equally corrupted.

The **Sobel edge-enhancement operator** requires the image to be convoluted with two templates.

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

These two images are then combined in two ways:

- The first combine edge magnitudes to estimate the **edge strength**:

$$E_{mag} = E_1^2 + E_2^2$$

- the second uses the ratio of the edge magnitudes to estimate **edge orientation**:

$$E_{orient} = \tan^{-1} \frac{E_1}{E_2}$$

Processing in the scale space provides a more efficient method of implementing many image recognition and feature detection operations. The region where the grey value changes, which will correspond to the **edge region**, a region that will contain the point we wish to label as the edge, will yield a finite result. **Location estimate** is at the centre pixel. It is more robust against noise

The **Canny/Deriche edge-detector** convolves image with Difference of Gaussian (DoG).

8.3 Global transforms

The **Hough transform** was first proposed as a means of detecting linear features in an image. Each point in the image that might contribute to the object is then transformed into the set of parameters corresponding to all of the objects it might contribute to. Having transformed all of the image points in this way, the sets of parameters are examined; those that have contributions from many points are assumed to be the parameters of objects that really are present in the image.

The accumulator is a simple array of counters used to record the number of contributions made to each parameter combination.

The Hough transform is used to locate lines in an image by firstly enhancing the edges in the image and then thresholding them.

8.4 Geometrical transformation

Linear transformations are arbitrary geometrical transformations moving a pixel at co-ordinates (x, y) to co-ordinates (x', y'). Examples of them are translation, rotation, scaling and shearing. These transformations may be applied in sequence and the resulting transformation will also be a linear one. Straight lines remain as straight lines and parallel lines remain parallel.

Nonlinear transformations introduce a degree of warping to the data. **Pincushion** result in the sides of the image being distorted less than the corners, while **barrel distortion** result in the sides of the image being distorted more than the corners.

When using the zoom to **reduce**, the new value of the pixel is weighted sum of nearest neighbours or it equals the nearest neighbour.
When using the zoom to **enlarge**, the new value of the pixel is weighted sum of nearest neighbours and you add noise to obscure pixelation.

Affine transforms preserve the length and the area, but not necessarily the angles. Examples of it are scaling, rotation and translation.

Image resampling may result in pixels being transformed to a non-integer output pixel. To solve this problem, we could truncate or round the co-ordinate values, creating holes in the image, or share the pixel's value amongst the neighbouring destination locations. However, the most elegant solution is to apply the transformation in reverse. In this case, we would take an output location and use the reverse transformation to determine where it originated in the input image. This location might still be non-integer, but we are able to guarantee that all output pixels will be populated. The output pixel's values are computed with nearest neighbour interpolation, which simply rounds (x, y) to the nearest neighbour, or **bilinear interpolation**, which takes a weighted sum of the four nearest neighbours to (x, y). This last one results in visually more pleasing output, specifically, the jagged effects generated by rotating straight edges are reduced.

9 Morphology

9.1 Thresholding

Thresholding is the process of reducing the grey scale of a monochrome image to two values. A **threshold value** aims to reduce misclassification error, assuming the two output values correspond to something sensible in the image. Any two different values are sufficient, 0 and 255 are often used as this maximises the contrast between the values.

$$g(x,y) = \begin{cases} 0 & f(x,y) < \theta \\ 1 & \text{otherwise} \end{cases}$$

The difficulty lies in selecting the correct value for the threshold. This is done by using the grey level histogram, which records the frequency of occurrence of each grey value in the image. In the ideal case, the threshold would clearly distinguish between contrasting object (foreground) and background. We may select some value in the trough between the two peaks as our threshold. However, we would not be able to identify which was which.

The threshold can be found manually, specific for the given image, (**modal method**), but also through **automated methods**.
A simple automated method of determining the threshold relies on knowing what proportion of the image is occupied by the foreground and whether the foreground is brighter or darker than the background. This method, called **P-tile**, simply defines the threshold as that grey level which selects the correct proportion of the grey levels.

Another method chooses the threshold to be the minimum between the histogram's peaks, aka the **mode** method.

An iterative method instead searches incrementally through the histogram for a threshold. Starting at the lower end of the histogram, we initialise a suggested threshold as the lowest value in the image; we compute the average of the pixels with grey values less than this threshold, call this L, and the average of the pixels with grey values greater than the suggested threshold,

call this G. We then compute the average of L and G. This value will be the threshold if it is equal to the suggested threshold, otherwise we increment the suggested threshold and repeat the process.

A second iterative method searches through the histogram more purposefully: the initial threshold value is suggested to be the average of the image's four corner pixels and the update value is equal to the average of L and G.

A problem arises when the local average in an image varies in a systematic manner across the image. Any threshold value that is computed using statistics derived from the whole image will therefore be correct in a few image locations, but incorrect over most of the image. The solution to this problem is to estimate the value of the threshold at each pixel, usually achieved by using a region surrounding the pixel.

When there are background pixels that are darker and brighter than the foreground, the solution is to define two threshold values. Since a colour image is represented using three colour channels, we must supply a threshold for each colour channel independently of the others and combine the results by conjunction. Since this method is not particularly flexible, there is a better one which thresholds the colour space according to each colour's distance from a user specified one. The number of erroneously classified pixels may be estimated by inspecting the histogram.

9.2 Processing binary images

The **structuring element** is a small template that is used to investigate the image. The structuring element's values, like the image it investigates, will be binary. Combinations of values making circular, square or cross-shaped templates are usually employed. The template will have an origin: it is this location that specifies the template's location in the image and therefore the position where the operation's result will be written. It is usual for the structuring element to have odd dimensions and the origin to be at the centre.

A structuring element is said to fit at an image location if all the image pixels that overlap the structuring element pixels of value 1 are also 1. The structuring element is said to **hit** at an image location if any of the image pixels that overlap the 1 values in the structuring element are 1.

Erosion: at each possible placement of the structuring element in the image, remove the pixel overlying the origin if the structuring element overlies a non-object pixel. Isolated object pixels are soon removed and the object itself shrinks at each iteration by an amount approximately equal to the size of the structuring element. The shrinkage is not only from the outside of the object but holes within the object are enlarged. It is applied to remove all unwanted small-scale structures in an image and to identify boundaries of objects. In fact, by subtracting a suitably eroded version of the image from the original, we identify those pixels that were removed by erosion.

Dilatation: the converse of erosion. At each possible placement of the structuring element in the image, add the pixel overlying the origin if the structuring element overlies an object pixel.

Erosion and dilatation have negative effect: the sizes of all objects are changed. Erosion and dilatation operators may be combined to enhance the positive effects and suppress the negative. **Opening** of an image is defined as erosion followed by dilatation using the same structuring element. **Closing** instead, is defined as dilatation followed by erosion. Repeated applications of the closing operator, as the opening one, using the same structuring element, will have no further effect on the image. Some uniformly textured images may be rendered smooth by repeated closing with progressively larger structuring elements.

The **top-hat transformation** is defined by the difference between an image and its opened version. **Granulometry** is the study of determining the size distributions of objects

9.3 Structural features

The **distance transform** is used to estimate the minimum distance between each point in an object and the background. To compute it, we use a 3×3 pixel structuring element to repeatedly erode an object until it is completely removed. At each iteration a set of pixels will be removed from the object. The iteration number is the distance transform for these pixels. The first iteration of the algorithm sets the pixels in the distance transform equal to the values of pixels in the original image (which will be zeroes and ones).

The **skeleton** of an object is a one pixel thick line that represents the object's shape. It must retain the following propertied:

- Connected image regions must thin to connected portions of the skeleton.
- The skeleton must be at least eight-connected.
- The locations of ends of lines must be maintained.
- The skeleton must approximate the medial line of the object.
- Extraneous spurs introduced by thinning must be minimised.

An object can be described as **convex** if a line connecting any two points on the object boundary lies entirely within the object. The **convex hull** of an arbitrary object is the smallest convex object that contains it. One method of computing it relies on tracing the outline of the object.

10 Region detection

Region detection, also known as **image segmentation**, is the process of dividing an image into separate and non-overlapping regions. A **region** can be specified by either defining the pixels that constitute it, or the pixels that bound it. Having segmented an image, every pixel in the image will have been assigned to a region, so the union of all regions is the original image. Colour values can be used effectively to isolate objects of known colour. Local thresholding methods are invoked in cases where the scene's illumination is spatially variant.

Since all pixels belong to a region, we may select any pixel and it will be part of some region. Since a region is made up of adjacent pixels, this pixel's neighbours are possibly part of the same region.

Algorithm 8 Region growing algorithm

while there are pixels in the image that have not been assigned to a region **do**
    Select an unassigned pixel and make it a new region. Initialise the region's properties
    if the properties of a neighbouring pixel are not inconsistent with the region **then**
        Add the pixel to the region
        Update the region's properties
        Repeat the if using the newly added pixel until no further neighbours can be added to the region

The properties of the region that we will maintain are the average grey value and the standard deviation of the grey value. The disadvantage with this method is that it takes time to execute.

An alternative to it is the **split and merge algorithm**. The splitting stage will apply the homogeneity predicate to a region of the image, initially the whole image is used. If the region is non-homogeneous, then it is split into quadrants and each examined by the predicate in turn. The

splitting will continue until either a region is homogeneous or it is too small for the homogeneity test to give sensible results. The homogeneity predicates that will be used are texture measures that are applied to areas of the image larger than a single pixel. Due to the deterministic nature of the splitting, uniform areas of the image may be represented by a number of adjacent regions, due to the deterministic nature of the splitting. A pair of regions will be merged if they are adjacent, they have similar grey scale or colour properties or the boundary between them is weak. A boundary is considered to be weak if it is of low contrast.

10.1 Edge detection

Edge detection enhances the edge structures in an image. The result of the process is an output image whose magnitudes are proportional to the likelihood of the pixel actually being part of an edge in the image.

Edge following is the process of tracking from one edge pixel to a neighbouring one and thus circumnavigating regions.

In many cases the object in the scene that produced the edge in the image has some regular structure: it may have a silhouette made of straight or regularly curved sections. A task that is sometimes performed is to replace a set of edge segments that approximate the real-world object, with a linear or curved segment that approximates the image data but is believed to be a truer representation of the real-world structure.

Edge segments may be accumulated into a linear approximation by aggregating segments and computing the best straight line that fits the data.

Segments may also be accumulated using the **hop-along** algorithm:

Algorithm 9 Hop-along algorithm

```
The next k edge segments are taken
A straight line is fitted between the first and last points
The distance between the line and each point is computed
if the maximum distance is above a threshold then
    The points up to the point of maximum distance become an approximated segment and
    the algorithm continues with the remaining points
if the maximum error is below the threshold then
    This approximation is compared with the previous one
    if they are collinear then
        They are merged
The next k edge segments are collected and the algorithm returns to fit a new line
```

10.2 Scale based methods

A **pyramid representation** of the image data will have at its base the original resolution data. Moving towards the apex of the pyramid, the resolution of the data will decrease, until, at the apex, we have the smallest resolution possible, which might be a single pixel or group of features.

The first method of generating the pyramid is **image averaging**. Pixels in a layer above the base layer are derived by averaging a set of pixels in the layer below. We cannot isolate an object using size information alone: if we select a scale suitable for an object and investigate that level of the pyramid, we shall also find information relating to objects of similar scales. The image

pyramid also requires more storage than the original data (almost twice as much). However, generating a pyramid does allow more rapid processing of data.

Applying a set of smoothing operators to the data may also generate a multiresolution version of an image. Manipulating the information is simpler, but there is much redundant information to be stored. The second one is performing **multiresolution feature extraction**, using some form of edge detector that includes a scale dependent parameter.

The **region map** is a method of recording the regions that have been found in the image. It is simply an image of region identities. We define a numerical identity for each region which is placed in the equivalent pixel of the region map as that pixel is assigned to a region. Since regions may also be defined by their outlines (boundaries), it is appropriate to use this information to delineate the regions of an image.

A **quadtree** is a tree structure in which each node has zero or four child nodes, which are themselves quadtrees. The root node will correspond to the whole image and the leaf nodes to uniform rectangular regions. The linkage between regions (made in the merge phase of the algorithm) cannot be represented directly in this scheme, it must be a separate piece of information.

11 Region description

A contiguous set of pixels that share some property and is **connected** (you can trace a path form one member to all others) is known as a **connected component**, **blob**. The process of identifying these components and labelling them is known as **connected component analysis**. An alternative method of identifying the pixels in a connected component uses a two pass algorithm, which works from left to right and top to bottom. In the first pass we are looking for pixels that could belong to a component. Each pixel is labelled with a numerical component label as it is found, the label it is given will be dependent on the labels of any previously labelled pixels. As we are searching systematically through the image, only those pixels in the line above and to the left of the current pixel can have been labelled. Three possibilities may occur:

1. None of the adjacent pixels has been labelled: the pixel is given the next free component label.

2. One or more of the adjacent pixels have been labelled but have the same label: then the pixel is given this label.

3. Two or more adjacent pixels have been labelled with differing labels: the current pixel is given one of the labels and the equivalence between the newly connected regions' labels is recorded.

After the first pass, some regions will have a consistent label, while others will have an inconsistent one. Thus, the second pass will simply relabel the inconsistent labels according to the equivalence table. A set of contiguous labels is not required but can be achieved by examination of the equivalence table.

The **area** of a region is simply the number of pixels making up the region. In order to compute it, we compute the histogram: each entry is equal to the number of pixels in that region.

The **perimeter** of a region is the total length of its outline.

A compactness measure, C, can be computed as $C = \frac{p^2}{A}$

11.1 Outline description

Descriptive information can be derived from blobs: moments of area, chain codes and colour distribution.

11.1.1 Moments of area

It is another method for deriving descriptive information from blobs.

$$M_{\alpha\beta} = \sum_{image} x^{\alpha} \cdot y^{\beta} \cdot f(x, y)$$

$f(x, y)$ is a binary function:

- $f(x, y) = 0$ if the pixels at (x, y) is outside the blob.
- $f(x, y) = 1$ if the pixels at (x, y) is inside the blob.

When $\alpha = 0$ and $\beta = 0$ and $f(x, y) = 1$, then one pixel will be added to the total area of the blob ($x^0 \times y^0 \times 1 \rightarrow 1 \times 1 \times 1 \rightarrow 1$).
Thus, for the normal case, when we are just interested in the blob's area, we set $\alpha = 0$ and $\beta = 0$. However, we can also compute other properties with the formula above. When $\alpha = 1$ and $\beta = 0$, we get the sum of the x values of the blob's pixels.

We can also compute the coordinates of the blob's centre of gravity:

$$(C_x, C_y) = \left(\frac{M_{\alpha=1,\beta=0}}{M_{\alpha=0,\beta=0}}, \frac{M_{\alpha=0,\beta=1}}{M_{\alpha=0,\beta=0}} \right)$$

The tuned version of the equation, which allows moving around the blob without causing the central moments to change, is:

$$M_{\alpha\beta} = \sum_{image} (x - \bar{x})^{\alpha} \cdot (y - \bar{y})^{\beta} \cdot f(x, y)$$

where $M_{\alpha\beta}$ is the central moments of area and $(\bar{x}, \bar{y})$ is the centre of gravity.

Moments can be defined for all values of α and β. There is a limit to the number of useful ones. We can use lower order moments to make higher order ones invariant to position, orientation and size of region. We can compute it for non-binary images. We can modify the computation to use labelled blobs. The values of the moments can be used to discriminate blobs based on size/shape.

11.1.2 Chain codes

After an image was processed by CCA, chain codes can help us find interesting properties of the connected components, including outline, perimeter and area. A **chain code representation** of a region will include the image co-ordinates of one point on the boundary of the region, plus a sequence of displacements that will take us from one pixel to the next connected one on the boundary and eventually return to the starting point.

The **boundary pixels** could be pixels inside the region that are adjacent to at least one background pixel, or they could be pixels in the background that are adjacent to at least one region pixel. Chain codes provide a position independent means of representing the outline of an object. It is also possible to modify the codes such that they become orientation independent by making the chain codes' directions relative to the tangent of the boundary instead of fixed to the

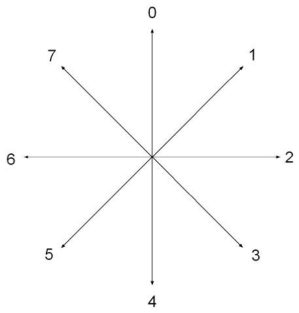

Figure 10: Chain codes

image's sides. Using eight connected codes, even numbered codes represent a distance equal to the separation between pixels. Odd numbered codes represent a distance equal to $\sqrt{2}$ times the pixel separation (derived from Pythagoras).
The total **perimeter** length is therefore the number of even numbered codes plus $\sqrt{2}$ times the number of odd numbered codes ($p = \#even + \sqrt{2}\#odd$).

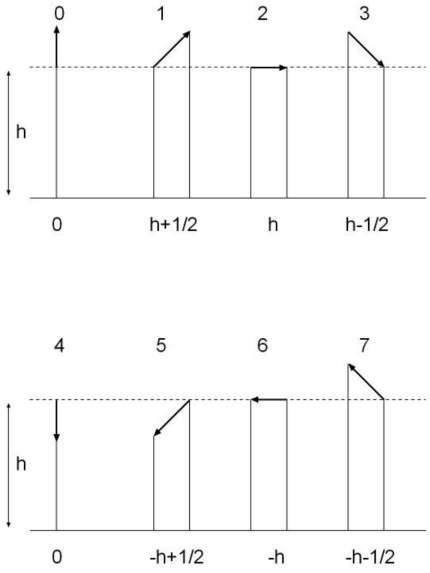

Figure 11: Computing area using chain codes

The **area** of a region is estimated by integrating the contributions due to each chain code. The **magnitude** of the area contributed by each codeword is the area of the quadrilateral bounded by the baseline, the displacement and the parallel lines joining the start and end points of the displacement to the baseline. h stands for the height, and it is not a constant.

The major disadvantage of this is that if the image is affected by noise, as it will be, then this will be at the pixel level and the chain codes will be directly affected. This difficulty can be avoided if we replace approximately linear segments of the boundary by straight line segments. The boundary of an object is thereby reduced to a polygon. This is achieved in two stages: In the first stage, sections of the boundary are divided into linear segments. In the second stage adjacent segments are merged, if necessary.

11.1.3 Others

Splines provide a compact representation method in that two end points and two intermediate control points represent one curve section. Adjacent curve sections will share end points to ensure continuity.

The **phi-s code** is a representation of an object's border that creates a plot of distances from a point to the border as a radius is rotated about the point. Recognition will be achieved by comparison of the curve with the curve derived from known objects.

Object boundaries can also be described as a sequence of structural building blocks that are extracted from the primitive boundary description elements. The strength of this approach is its immunity to scale, rotation and translation: no transformation will have any effect on the description, beyond changing its start point.

12 Region labelling

Region labelling is the process of associating a label with a region of the image. Four major factors make it a difficult problem:

- The natural variability of the scene will cause an equal variability in the captured images.
- An image is a two-dimensional projection of a three-dimensional object.
- Systems are often designed to recognise one object and make measurements of it.
- Occlusion: as the number of objects in the scene increases, so does the probability of one of them partially obscuring another.

12.1 Object labelling

The simplest method of labelling an object is to compare it against a previously identified sample.

Template matching is the process of comparing regions of an image and a template. A measure of the similarity or dissimilarity between object and template is computed, we can therefore find the locations where the template matches a region of the image.

Three ways of computing the dissimilarity have been proposed:

- Compute the maximum absolute difference between template and the region of the image it overlies.
- Compute the maximum absolute difference between template and the region of the image it overlies.
- Compute the sum of squared differences. It is the preferred one as it is the easiest to compute.

Although **cross-correlation**, or template matching, is extremely effective when used correctly, we must be aware of its severe limitations. A specific template is required for each object to be recognised and the matching fails catastrophically if the objects are partially occluded.

13 Appendix: OpenGL

It is a **specification** of an Application Programmer's Interface: a set of functions for 3D graphics. It has a fixed pipeline: programming is simpler and there is more built-in support, but there is a lack of flexibility and it cannot fully exploit the GPU.

A key feature of the design of OpenGL is the separation of interaction (input and windowing functions) from rendering. OpenGL itself is concerned only with graphics rendering.

OpenGL only generates pixels. Its main features are: 3D graphics, coordinates transformations, a camera for viewing, hidden surface removal, lighting and shading, texturing and pixel operations. To handle interaction devices, we use the additional library **GLUT**.

OpenGL mantains two transformation matrices internally:

- **Modelview**: used for transforming the geometry and specifying the camera.
- **Projection matrix**: controls the way the camera image is projected onto the screen.

Every 3D point you ask OpenGL to draw is automatically transformed by these two matrices before it is drawn. When we call a transformation function, it creates a corresponding temporary matrix and multiplies the modelview by it before discarding it.

The order is the reverse of how we would write it down logically.