National Centre for Computer Animation

Sprite Engines 1

To study and not think is a waste.
To think and not study is dangerous.

ES Overview

- Introduction to ES
- **2D Graphics in Entertainment Systems**
- Sound, Speech & Music
- 3D Graphics in Entertainment Systems

Sprite Engines

- The Framebuffer
- Sprites
  - Definition
  - Blitting
- Advanced Concepts
  - Colour Keys and Blending
  - Drawing Order
  - Collision Detection

Framebuffer

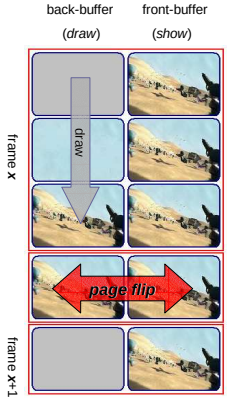
A Framebuffer is an area of grahics (*video*) memory that can be drawn to which can hold the graphics information for a single frame.

- It is the rendering context containing - amongst others - the data for the pixels that need to be drawn.
  (*in the pixel type used - usually only colour information but can also contain an alpha channel*)
  - This information is usually stored in the "display mode" of the framebuffer (*including video resolution and colour depth*).
- On modern graphics hardware the framebuffer is usually located in the graphics card's video memory.

Double-Buffering

- allows whole scenes to be rendered before being displayed
- reduces flickering & artefacts (*whole scene is changed at a vertical refresh*)

1. a scene is renderd into a second (*invisible*) framebuffer (*back-buffer*) which is an extension of the framebuffer
2. once the rendering has finished the back-buffer is exchanged for the currently displayed framebuffer (*front-buffer*) – this process is called page flipping
   – page flipping must happen once a frame has been displayed
3. the old front-buffer is now the back-buffer onto which the scene is rendered



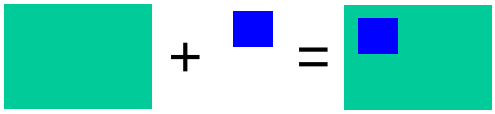
Sprites

- A sprite is a small 2D image that is drawn onto an existing scene to create a more complex scene.
- In early entertainment systems sprites were realised in hardware.
- New systems just handle them as images that are copied onto the framebuffer.
- Sprites can create the illusion of dimension by scaling down sprites that are supposed to be farther away from the viewer.

Blitting

Blitting or “bit blitting” is a computer graphics operation in which two bitmaps are combined into a new one.

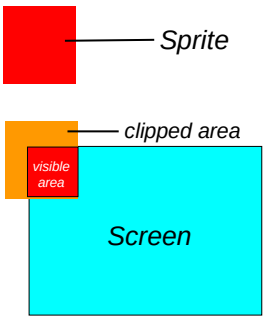
- This is usually done on the frame buffer
- The rendering of sprites onto a frame buffer is usually done using a blit operation.
- The name derives from the BitBLT machine instruction for the Xerox Alto computer, standing for "Bit Block Transfer". (*Wikipedia*)



Clipping

Clipping of an image or a sprite means to only draw those bits that will be visible to the viewer.

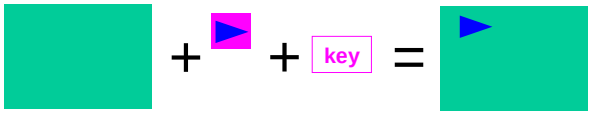
- Clipping is not a trivial operation (*especially in 3D*) but often simplified by library routines or hardware operations.



Colour Keying

A **colour key** is a specially selected pixel colour which will be replaced with a transparent pixel if it is blitted onto a framebuffer as a sprite.

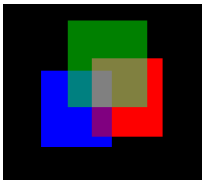
- This is similar to chroma keying which is the technology used in TV and film to achieve blue-screen or green-screen effects.



Alpha Blending

Alpha Blending is a per-pixel operation performed on the framebuffer.

- done using the Alpha values of RGBA colours
- useful for creating transparency or for compositing
- combining the colour of what is being drawn (*source*) with that of the respective pixels that already exist in the framebuffer (*destination*)



alpha values can usually be understood as opacity in percent (*clamped between 0 and 1*):

- alpha value of 0.0 = 0% opacity = fully transparent
- alpha value of 1.0 = 100% opacity = fully opaque (*solid*)

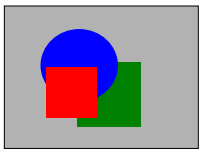
Otherwise:

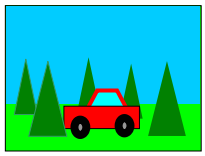
- alpha value of 0 = 0% opacity = fully transparent
- alpha value of 255 = 100% opacity = fully opaque (*solid*)

Drawing Order

The Painter‘s algorithm (*also called priority fill*) means that elements are drawn on top of each other (*like a painter on a canvas*).

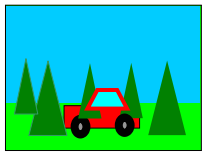
- It can be used to create the illusion of depth in a scene if more distant objects are drawn smaller behind the front images.





Drawing Order

- If Images overlap or more distant objects are drawn later than close objects, the painter’s algorithm can fail:



- The solution is **Z-ordering**, i.e. ordering objects by distance from the viewer and then drawing them from back to front.

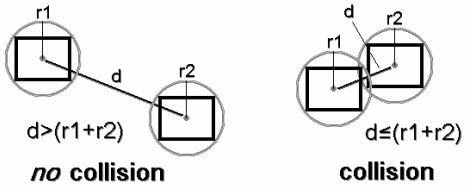
Collision Detection

- collision detection is useful for games & dynamics
- in its most simple form collision detection is done using a bounding sphere around an object
- to do that the radius for the collision sphere has to be stored for each object in a scene

note: collision detection only registers the collisions between objects. To use this data in a meaningful way requires collision resolution.

Collision Detection

a collision between two objects has occurred when the distance between the centre points of two objects is less than or equal to the sum of the radii of their bounding spheres



a collision between an object and a plane has occurred when the distance between the centre point of the object and the plane is less than or equal to the radius of its bounding sphere
because the distance calculation uses a square root, this case can be optimised by calculating the square value of the radii of the bounding spheres to use to compare with the square of the distance (*without calculating the root*)

Remember:

Using Pythagoras we can calculate the distance d between 2 points a & b: $d = \sqrt{((b_x-a_x)^2 + (b_y-a_y)^2)}$