Particle Systems

Particles and rendering transparency

Programming – Computer Graphics

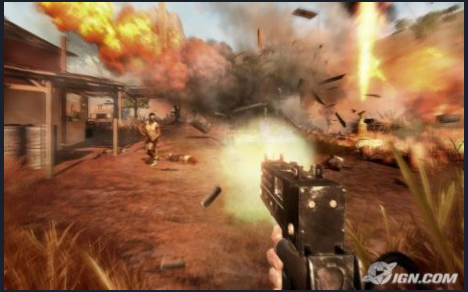
aie
SPECIALIST EDUCATORS IN
GAMES, ANIMATION & FILM VFX

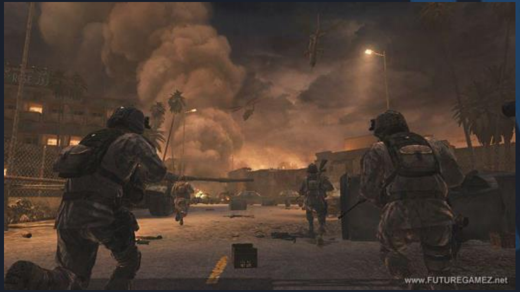
Contents

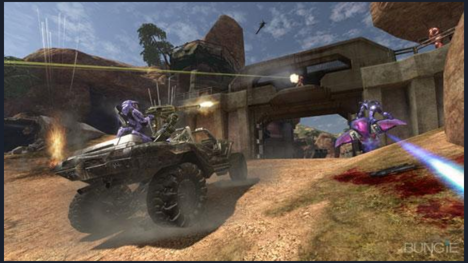
- Particle Effects
- Particles
- Emitters
- Particle Systems
- Rendering Particles
  - Billboarding
  - Alpha Blending

Particle Effects

- Particle effects are a type of procedural geometry that are used to create specific types of special effects in games:
  - Fire / Smoke
  - Clouds
  - Dust
  - Rain
  - Splatter
  - Trails
  - Fireworks
  - Lasers / Muzzle Flash



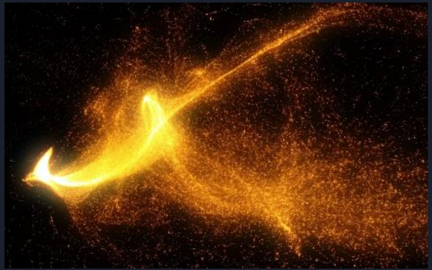


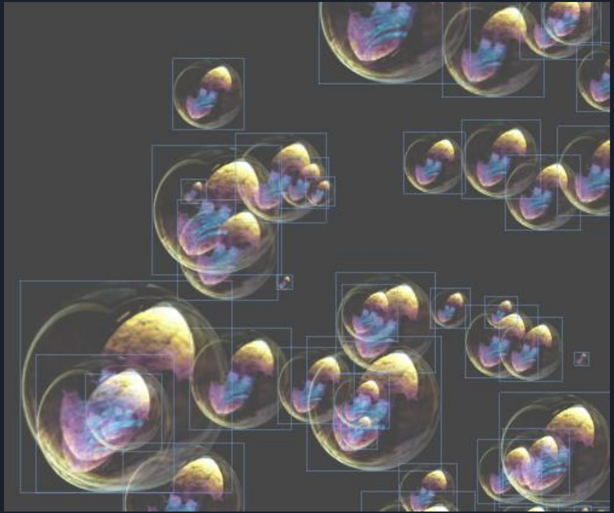


aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Particles

- Particle effects are usually represented by an array of objects which contain information such as:
  - Position
  - Rotation
  - Transparency
  - Colour
  - Size





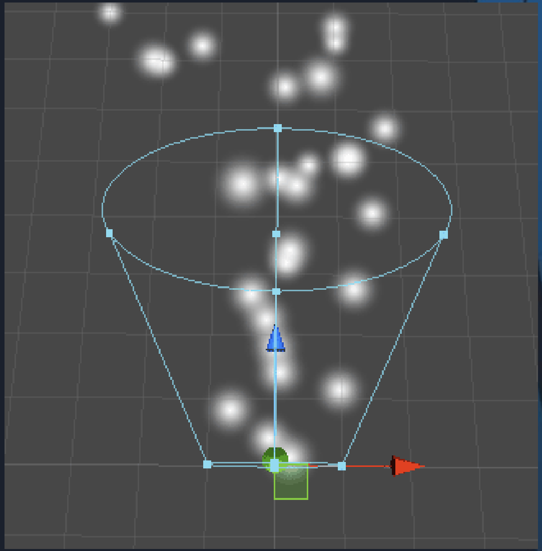
- Each particle is usually rendered as a billboarded quad called a Sprite

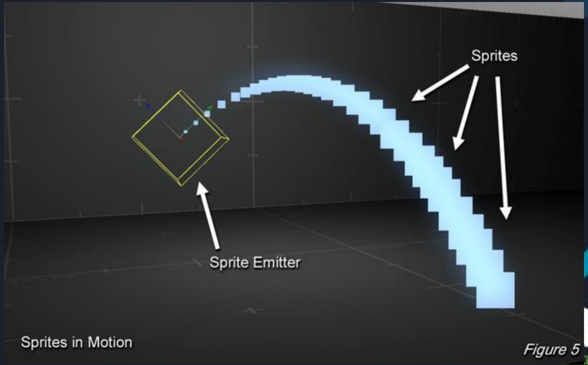
Particles

- Each particle also has information about how it changes over time:
  - Velocity
  - Drag
  - Change in Rotation
  - Change in Transparency
  - Change in Size
  - Change in Colour
  - Lifespan

Emitter

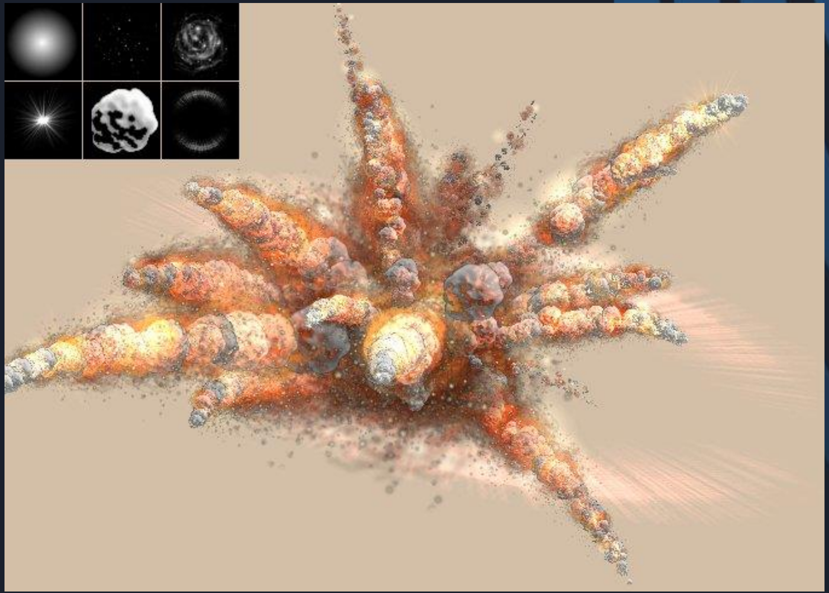
- Particles are spawned from an Emitter
- Emitters can be any shape, with particles spawning from within and / or on it
  - Point, cone, box, sphere, line, and mesh are common emitters
- Usually specifies starting position and velocity of particles
- Particles spawn from the emitter at specified rates, with a containing system maintaining the collection of alive particles





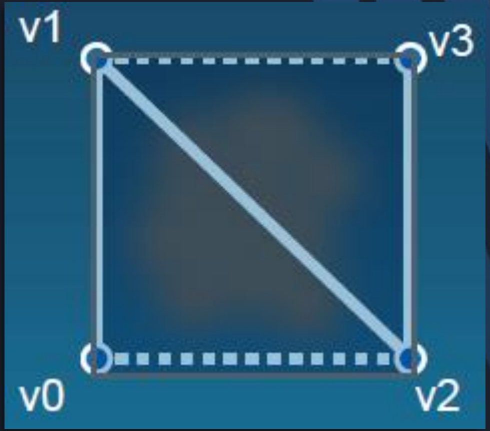
Particle Systems

- A parent system updates the collection of particles each frame
- Particle lifetime is reduced, and particles that die are no longer rendered
- Emitters typically have a spawn rate and spawn new particles each frame as needed
- Different emitters can be combined, each spawning particles with different properties, to create complex particle effects
  - Such as a smoke emitter + flame emitter + shockwave emitter



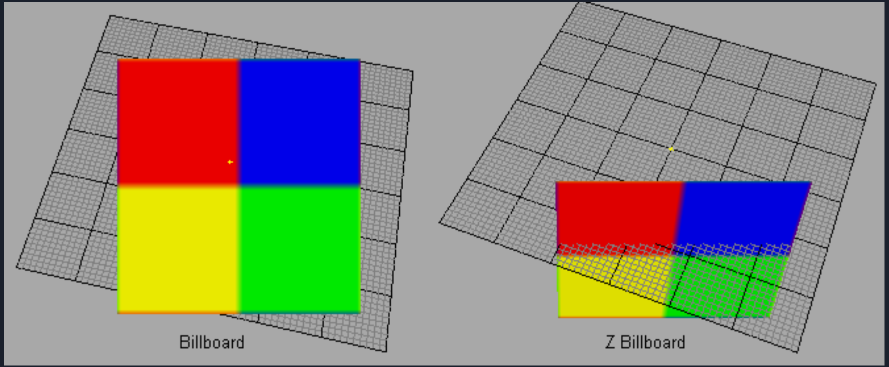
Rendering 3D Particles

- In order to draw a particle, a quad of two triangles (i.e. 4 points) which always face the camera, is drawn
  - We render all of the visible particles
- Making them face the camera is called Billboarding
- For most effects we also enable alpha blending
- We also typically disable writing to the depth buffer when rendering particles
  - We still use depth testing however
  - This is because we don't know the order that our particles will render in and a closer particle might render before a further away particle and obscure it



Billboarding

- Billboarding is the term given to orientating an object, usually a quad, to face the camera
  - We can constrain axis of the orientation if we want an object to face the camera but be orientated to a set axis

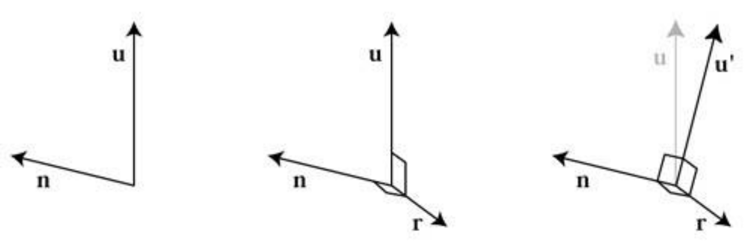


Non-constrained Constrained

Billboarding

- To orientate an item towards the camera we create a transform facing the camera:
  - We can treat the axis of the transform as right, up and forward, which would represent the X-Axis, Y-Axis, and Z-Axis of a matrix
  - The forward axis of the transform is simply a unit vector from the item towards the camera
  - The right axis of the transform is the result of a cross product between a temporary “up” axis, usually (0,1,0), and our forward axis
  - The up axis of the transform is simply the result of a cross product between our forward axis and right axis

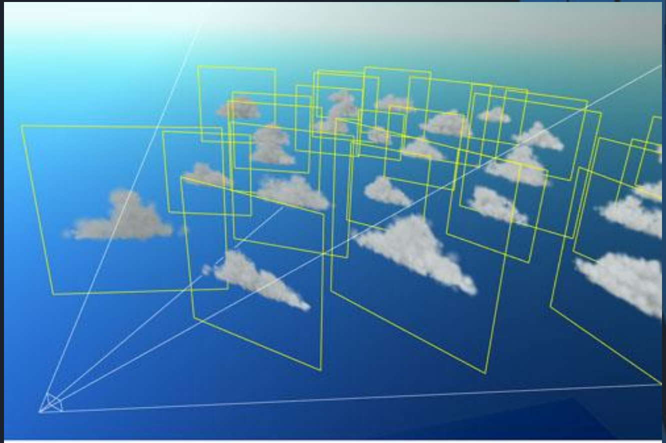
- This transform is then used to orientate the four corners of an axis-aligned quad



n is a vector towards the camera

Billboarding

- We can also lock an axis of the sprite, for example, locking an object’s up axis:
  - Create a temporary “forward” vector from the object to the camera
  - The right axis of the transform is the result of a cross product between the locked up vector and the temporary “forward” axis
  - The forward axis is the result of a cross product between the right axis and the locked up axis
  - We leave the up axis as the locked “up” axis
- This technique is useful for items in the distance, such as clouds

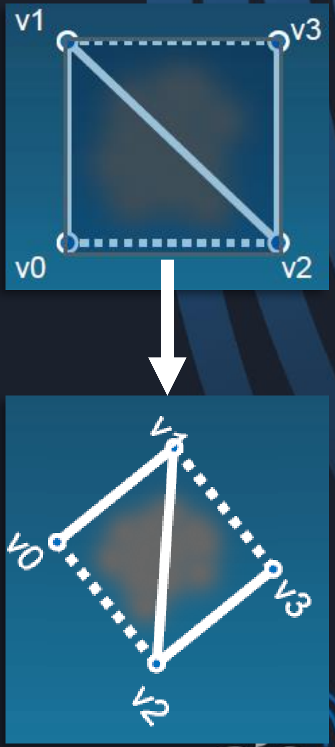


Rotating Billboards

- For rotating particles we calculate the quads corners and rotate them before applying the billboard transform

- The size of the particle can be represented by a scale in the transform, or by scaling the corners before billboarding

- We then just transform the 4 points of the quad by our new transform to orientate the particle correctly
  - Repeat for all particles



Alpha Blending and Depth Testing

- Once orientated, the particles usually need to be rendered with Alpha Blending **enabled** and Depth Write **disabled** while still using Depth Testing
  - Alpha Blending is the term for blending colours based off alpha channel values
  - Depth Write is the term for writing a pixel's depth to the depth buffer, which can be optionally disabled
  - Depth Testing is the term for testing a pixel's depth to determine if it is occluded by a closer pixel
    - Usually any pixel further away than the current pixel at the same location is culled
    - This can be disabled, and different comparison methods can be used

Alpha

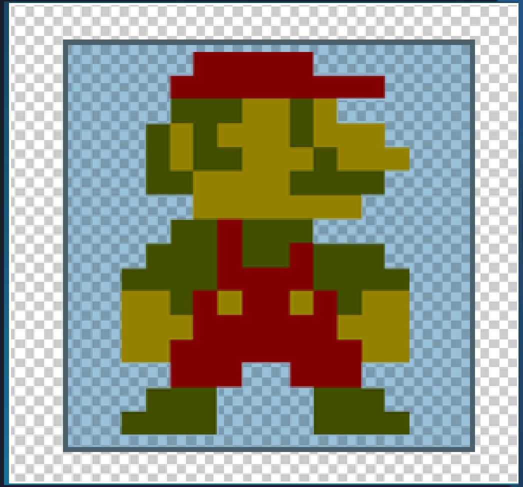
- Transparency is usually dictated by an image's Alpha Channel

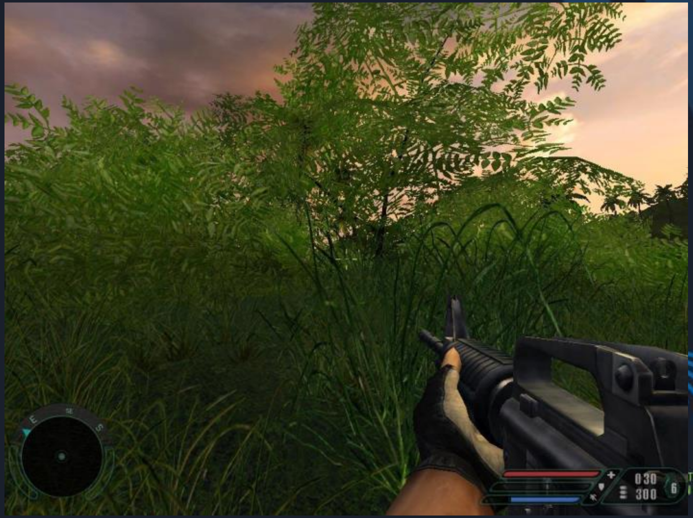
- There are two types of Alpha
  - Alpha Test: boolean visible or not-visible based on Alpha comparison
  - Alpha Blend: interpolated blending of original colour and new colour, based off the Alpha value

Alpha Test

- Pixels can either be fully transparent or opaque
  - As in there is no semi-transparent)
  - Also called 1-bit Alpha

- A fast technique usually used to define hard edges at a pixel level
  - Common for grass and leaves in games
  - 8-bit style 2D games

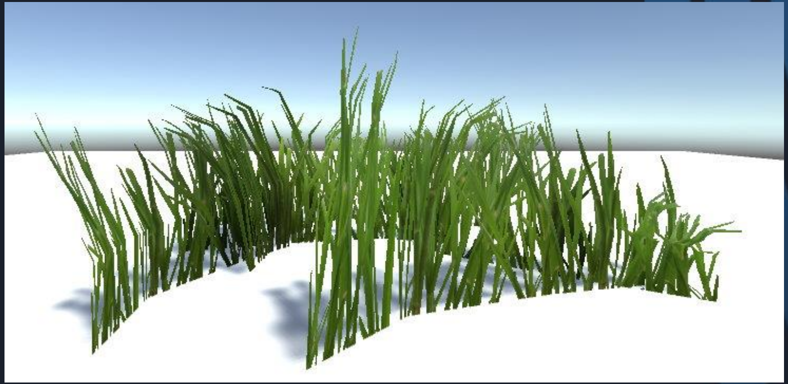




Alpha Testing

- The Alpha Test consists of two parts
  - The Alpha Test Value
  - The Alpha Test Function

- The Alpha Test Function is a simple comparison function for determining if a pixel is culled or not
  - It has the following values:
    - EQUAL
    - GREATER
    - GREATER_EQUAL
    - NEVER
    - NOT_EQUAL
    - LESS
    - LESS_EQUAL
    - ALWAYS



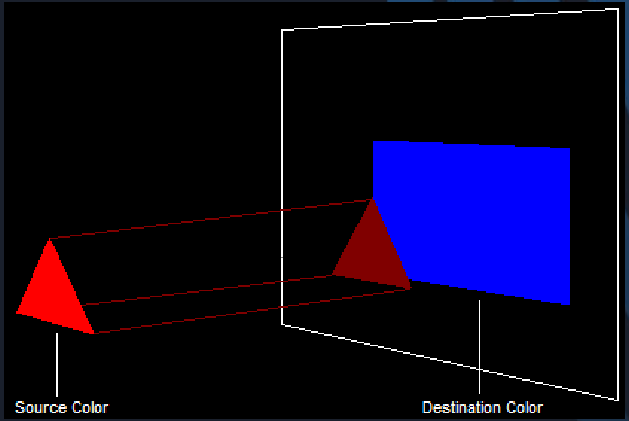
- The Alpha Test Value is simply a number to compare the pixel's Alpha value to
  - i.e. a pixel with an Alpha value of 0.7 compared against an Alpha Test Value of 0.5 with an Alpha Test Function of GREATER_EQUAL would pass, and thus be visible

Alpha Blending Options

- When you wish to interpolate pixel colours based on transparency, which is common with particles, then you need to enable Alpha Blending

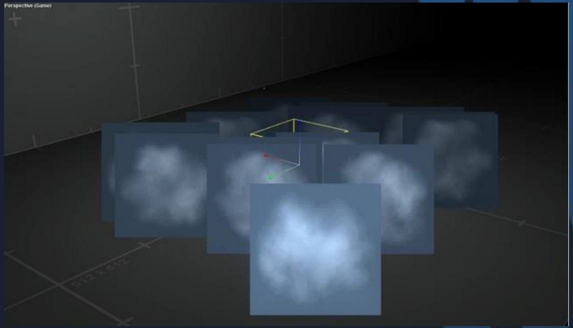
- Alpha Blending specifies how to combine a new pixel colour, the Source, with one that already exists in the pixel it is rendering into, the Destination
  - Occurs after the Fragment Shader

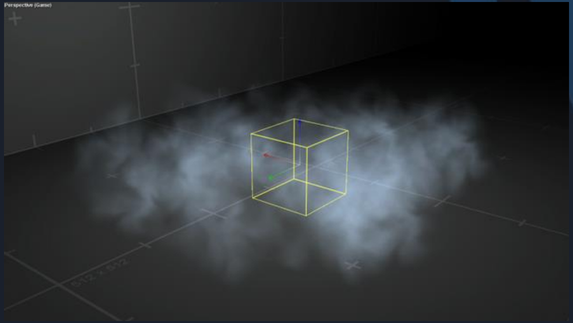
- There are two parts to Alpha Blending
  - Blend Equation
  - Blend Function



Alpha Blending Options

- The Blend Equation specifies how to combine the two pixels
  - Options are Add, Subtract, Reverse Subtract, Min and Max
  - The default is Add
- The Blend Function is mathematical parameters applied to the Source and Destination first
  - There are many parameters, such as Zero, One, Source Alpha, One Minus Source
    Alpha, and more
- The blend usually works as:



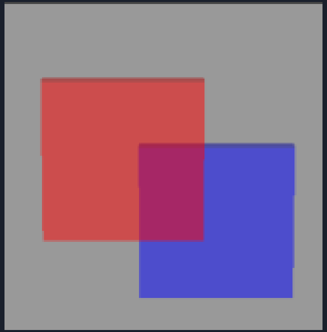


$$equation(\,source\ colour\ \times source\ parameter, dest\ colour\ \times dest\ parameter\,)$$

Standard Alpha Blending

- Standard Alpha Blending uses the following options:
  - Equation: Add
  - Source Blend Mode = Source Alpha
  - Destination Blend Mode = Inverse Source Alpha

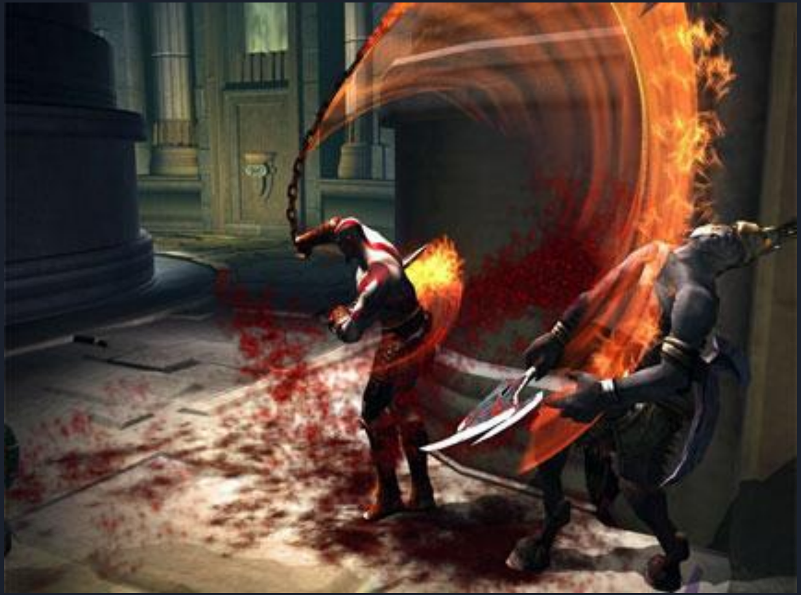
- For example:
  - Destination Colour = Blue
  - Source Colour = Red
  - Source Alpha = 0.75
  - FinalColour = (0.75 * Red) + ((1 – 0.75) * Blue)



Standard Alpha Blending

- Standard Alpha Blending is good for adding a transparent colour on top of what is already in the scene

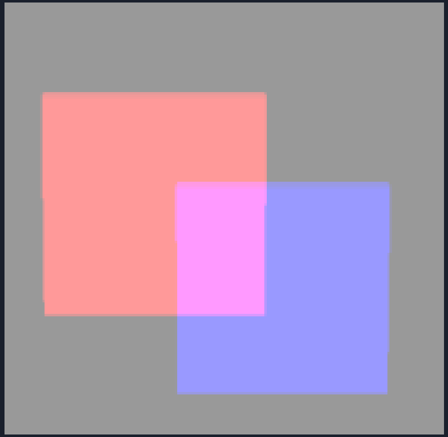
- For Example:
  - Smoke
  - Blood splatters
  - Dirty glass in windows
  - Water surface effects



Additive Alpha Blending

- Additive Alpha Blending commonly uses the following modes:
  - Equation: Add
  - Source Blend Mode = Source Alpha or One
  - Destination Blend Mode = One

- For example:
  - Destination Colour = Blue
  - Source Colour = Red
  - Source Alpha = 0.75
  - FinalColour = (0.75 * Red) + (1 * Blue)



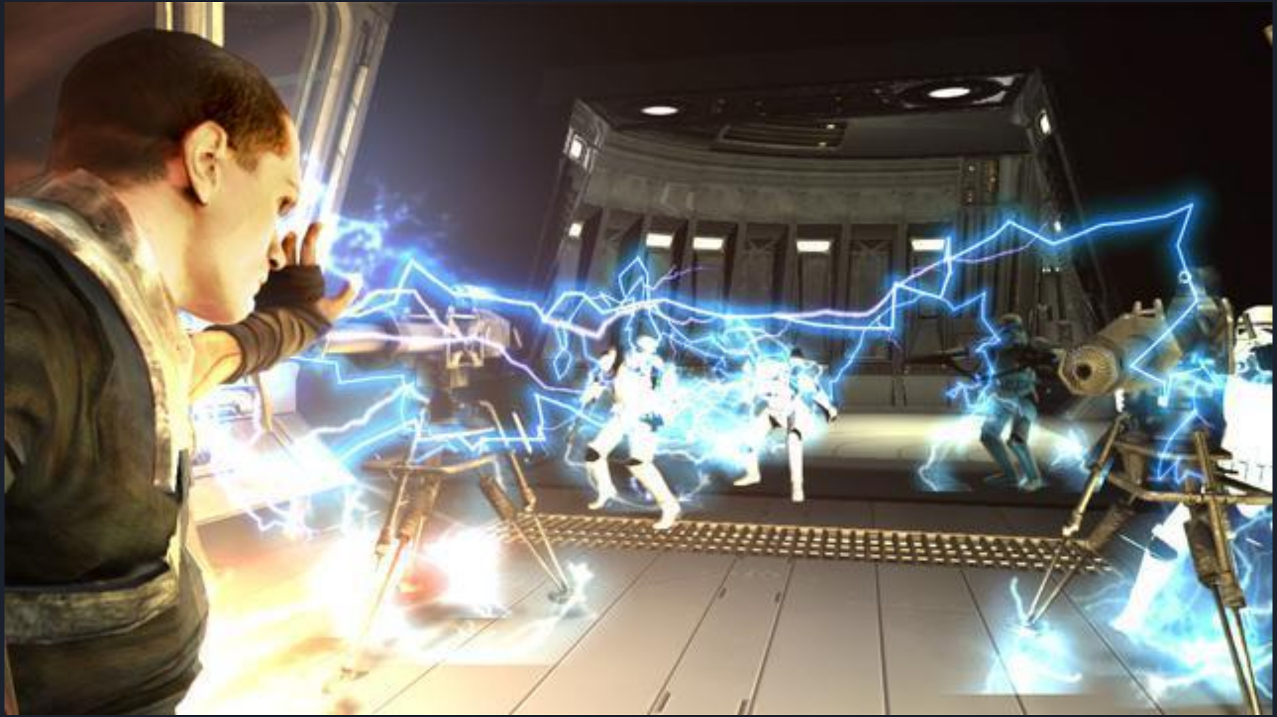
Additive Alpha Blending

- Additive Alpha Blending is good for creating effects that look over brightened, or "hot"

- For Example:
  - Bright Sun Glare
  - Fires
  - Explosions
  - Lens Flares
  - Lightning Strikes
  - Weapon Effects

AIE SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

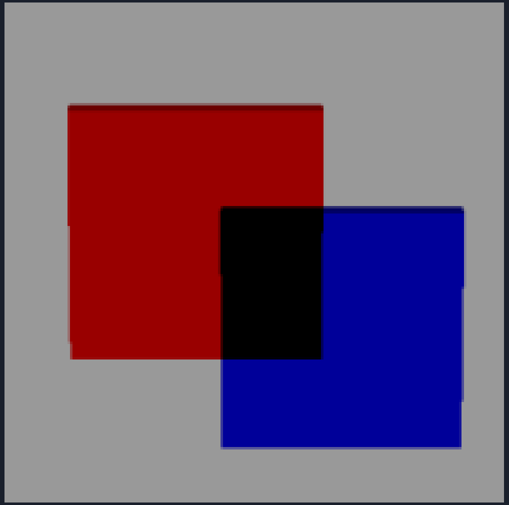
Additive Blending



Subtractive Alpha Blending

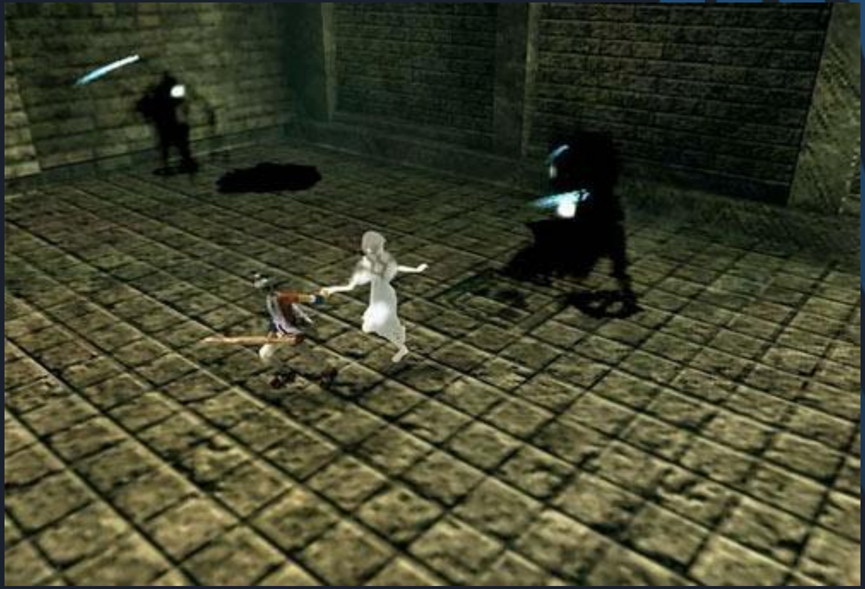
- Subtractive Alpha Blending use the blend modes:
  - Equation: Subtract
  - Source Blend Mode = Source Alpha
  - Destination Blend Mode = One

- For example:
  - Destination Colour = Blue
  - Source Colour = Red
  - Source Alpha = 0.75
  - FinalColour = (1 * Blue) – (0.75 * Red)



Subtractive Alpha Blending

- Subtractive Alpha Blending makes things look darker
- Not many real world applications but great for SFX in fantasy and Sci-fi



SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

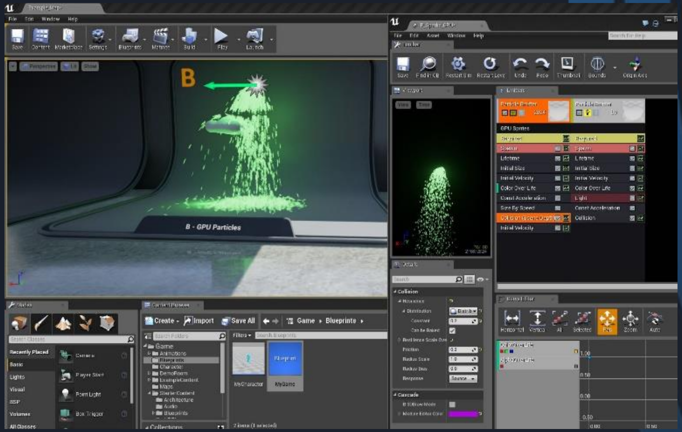
Depth Write and Depth Test

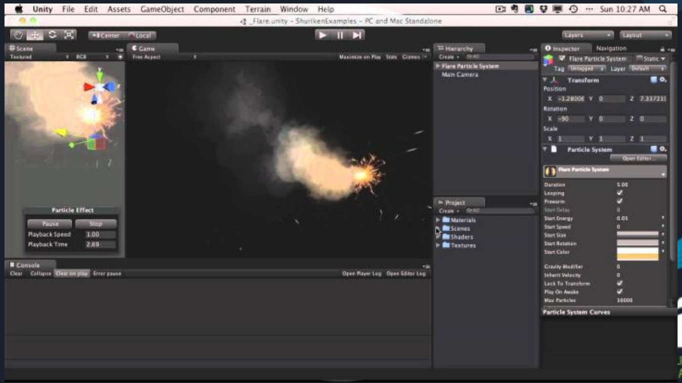
- The final step when rendering particles is to do with the Depth Buffer
- For blending to work correctly we need to render the particles last
- We allow them to test against the existing Depth Buffer so that particles behind solid objects don't render
- We also disable particles writing to the Depth Buffer
  - This is because we are unable to sort the particles correctly for blending, due to performance issues, so generally we don't allow them to write to the Depth Buffer
- Both Depth Writing and Depth Testing can be enabled and disable
  - Depth Testing also has similar comparison testing to Alpha Testing

Game Engines and Particles

- Most game engines have an in-built editor for creating Particle Effects
  - Effects are typically created by a designer or artist

- In code, effects can be triggered to play
  - Looping possible, depending on the effect





Summary

- Particles and Emitters can be combined to create interesting Particle Effects for games
  - Using the same code many different effects can be achieved just by changing the Emitter properties and Particle properties
    - The same code for a shockwave blast is used for snow falling or blood bursts

- Rendering 3D particles requires various blend states and Depth Buffer states
  - Plays a major role in the final visual appearance of particle effects

Further Reading

- Wolff, D, 2013, *OpenGL 4 Shading Language Cookbook*, 2nd Edition, PACKT Publishing

- Akenine-Möller, T, Haines, E, 2008, *Real-Time Rendering*, 3rd Edition, A.K. Peters