2D

GRAPHICS & JAVA

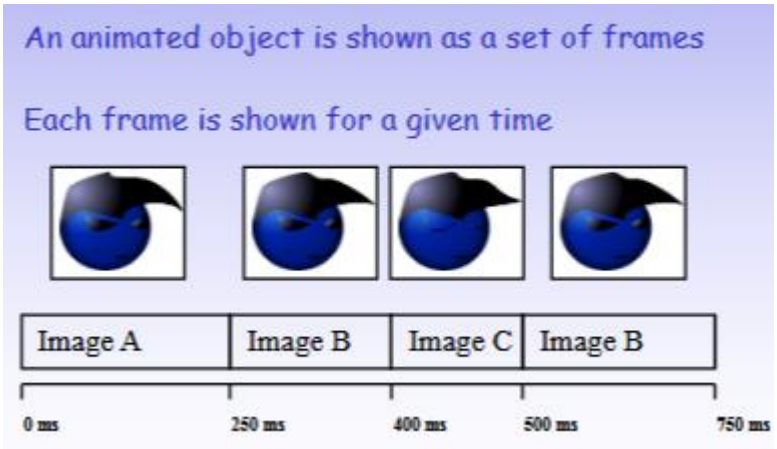
1. What things do we need to render graphics in Java?
    - A window object – i.e. a canvas to draw on
    - A DisplayMode object – defines type of display we will be drawing on, includes screen resolution, build depth, refresh rate
    - A graphics device – acts as virtual boundary to the graphics card

2. Where should we preload images, and why should we do it?
   Preloading images saves constantly reloading and refreshing the image, should preload in the go() method and redraw in the paint() method.

3. What is **animation**?
   An animation is a set of image frames, each frame shown for a given time.

4. What is a **sprite**?
   A sprite is an animated object, with a position, velocity, and other features.

5. How do we implement animation in Java?
    - Create an Animation class to control display of the relevant frame at the given time

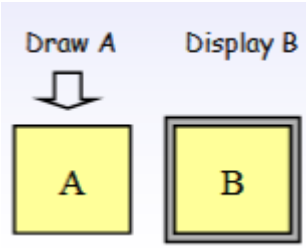


6. What are some key components of the **animation** class?
    - Animation Class contains:
        - A set of animation frames – AnimFrame
        - Index of the current frame
        - Current animation time
        - Total time the animation takes to render
        - Methods to render the animation

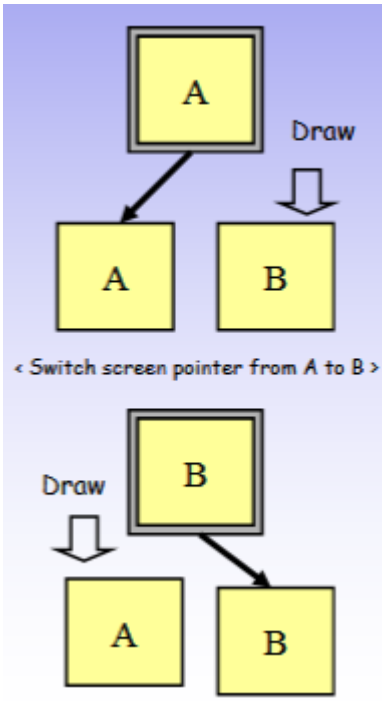
7. What is **flicker**, and why does it happen?
   Flickering occurs due to continuous drawing of the background and then the foreground image, meaning that the foreground image will constantly disappear and reappear in quick succession – can be fixed with double buffering.

8. What is **double buffering**? How does it work?

Render the complete image off screen to a 'buffer', and then copy the 'buffer' to the screen in one go, to eliminate the flickering issue. The buffered image is the same size as the screen, i.e. if the screen resolution is 1024 x 768 with 24 bits per pixel then we need an image of 2.25MB to be copied for each frame (1024 x 768 x 3).



9. What is **page flipping**? How does it work?
   Page flipping is like double buffering:
   - Image is drawn to an off-screen buffer
   - Instead of copying the entire buffer, the display is pointed to the buffer that it should use
   - Program 'flips' the pointers between each rendered frame
   - Executes considerably faster since no copying of large memory areas required



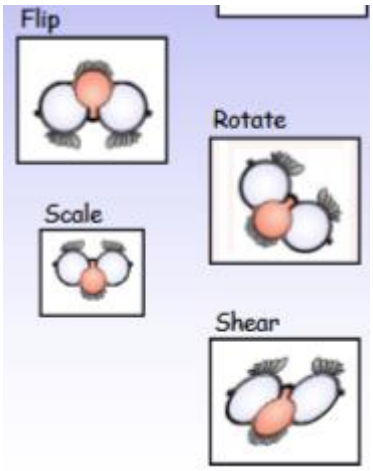
10. What is **tearing**, and why does it happen?
    Tearing occurs when a buffer is flipped or copied while the screen is being refreshed, resulting in part of the screen showing the old buffer and the rest showing the new buffer, hence causing a tear. Can be fixed by ensuring that the buffer is flipped just before the screen is refreshed (this is device dependent though).

11. Describe some strategies that can be used to avoid using **too much memory** in a 2D game.
    - Keep as much out of the update() method as you can
    - Only draw what is on the screen
    - Consider using buffers

SPRITES & TRANSFORMS

1. Why should we keep **draw()** out of the update method?
   We do not want to keep redrawing every single item to the screen every frame if we do not have to; lots of things can be omitted or simply moved around the screen – redrawing involves removing and then redrawing.

2. Name some types of **image transforms** we can do.
   - Translate – move image within its frame
   - Flip – flip about an axis, i.e. horizontal
   - Rotate – spin around a point
   - Scale – magnify or shrink to a different size
   - Shear – basically italics but for images

3. What **class** can be used to perform image transforms?
   The AffineTransform class.

4. What are the **3 main properties** of a sprite?
   - Animation – set of animation frames/durations
   - Position – X & Y co-ordinates on screen
   - Velocity – A speed and direction OR horizontal & vertical sped (in pixels/millisecond)

5. Describe the process of using this class.
   An AffineTransform is implemented as a 3x3 matrix of values or functions that are applied to a given x,y co-ordinate:

   [x'] [m00 m01 m02] [x] [m00*x + m01*y + m02*1]
   [y'] =[m10 m11 m12] [y] = [m10*x + m11*y + m12*1]
   [1] [0 0 1] [1] [0*x + 0*y + 1*1]

   They effectively define how much each original x and y point should be adjusted to implement the required transform.

USER INTERFACE – EVENTS

1. What is an **event listener**?

An event listener listens for each type of event that could be of interest to us.

2. Give some **examples** of event listeners.
    - KeyListener – listen for keyboard generated events
    - MouseListener – general mouse events such as a click
    - MouseMotionListener – mouse movement events
    - MouseWheelListener – mouse wheel movements

3. Describe the **purpose** and **components** of the GameCore class.
   GameCore is used for inheritance by being the main, centralised class with elements that other classes could use:
    - An abstract class with similar elements as before
    - An init method which sets up the display using the ScreenManager class, with options for full or windowed mode
    - A gameLoop method (like the animation loop)
    - An update method which we can override
    - A draw method which we **must** override
    - A stop method to stop the game loop
    - A useful loadImage method to load images

PLATFORM GAMES

1. Describe the process for drawing to the game world.

2. What is a **tilemap**?
   A file that tells the game how to display tiles in the game world, in our case made with notepad; each character is translated into a single tile in the game world.

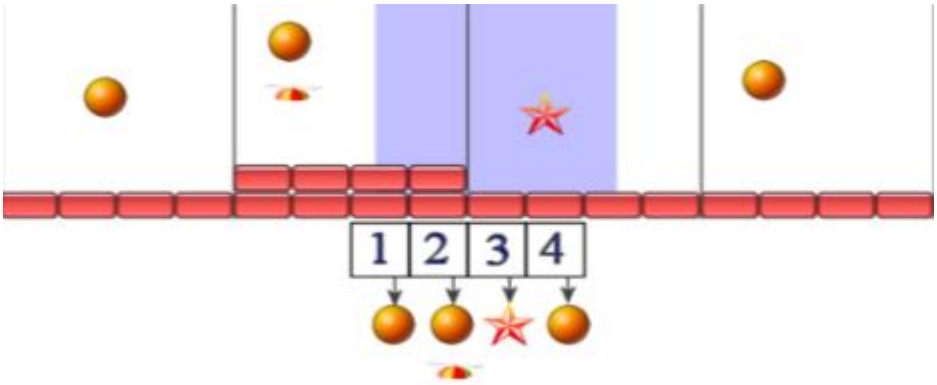
3. Write an example of a **tilemap file**.

```
64 22 32 32
// The first line should contain the width and height of the
// map and the width and height of each tile. A list of character to
// tile mappings is then provided where each character is preceded by a
// # character. The dot character always defaults to a blank space
// Note that the referenced files should be in the same directory as the
// tile map.
#*=dot.png
#h=heart_pickup.png
#l=lava.png
#g=ground.png
#u=underground.png
#q=left_platform.png
#w=left_wall_bottom.png
#s=left_wall_top.png
#p=right_platform.png
#o=right_wall_bottom.png
#k=right_wall_top.png
#1=greenGem.png
#2=redGem.png
#3=blueGem.png
#4=purpleGem.png
#5=whiteGem.png
// The actual tile map is preceded by the #map line
#map
uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu
o.....................................................................w
o.....................................................................w
o..................252.......qp.............................w
o........qp...qgggggggggggp........qggp.....................w
o...qp.....................................................w
o.....................................................11111w
o.......qgp.............................................1ggggggu
o.............qp.....................................1guuuuuu
o.............................................11...1guuuuuuu
o...11111..gg...ggg11.........................1.gg..ggguuuuuuuu
ugggggggggguullluuugg11...22..22.............1.gglllluuuuuuuuu
uuuuuuuuuuulllllluuugg111..11..11.........1.gguuuuuuuuuuuuuuuu
uuuuuuuuuuuuluuuluuuuuuggglggllgg111...1.1ggglluuuuuuuuuuuuuuuu
uuuu..uuuuuuluuuuuuuluuuuuuuuuuuuggk..sgggguuuuuuuuuuuuuuuuuuuu
uuu...uuuuuuluuuuuuuluuuuuuuuuuuuuuo..wuuuuuuuuuuuuuuuuuu.....
uuuu..uuuuuuuuuuuuuluu.............wuuuuuuuuuuuuuuuuuuuu....
uuuu..uuuuuuuuuuuuuluuh33...........wuuuuuuuuuuuuuuuuuuuu...
uuuu..uuuuuuuuuuuullluuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu
uuuuu..uuuuuuuuuuluuuuuuuuuuuuuuuuuuuu.....uuuuuuuuuuuuuuuuuu
uuuuuu..uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu
uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu
```

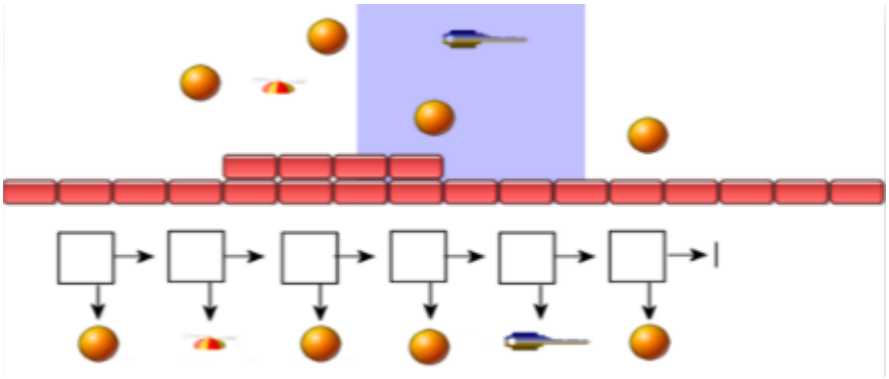
DRAWING SPRITES

1. Why is it best to do background images as **sprites** rather than images?
   Gives a good sense of motion if we speed up and slow down the background image using parallax scrolling – easier to do if we make it a sprite.

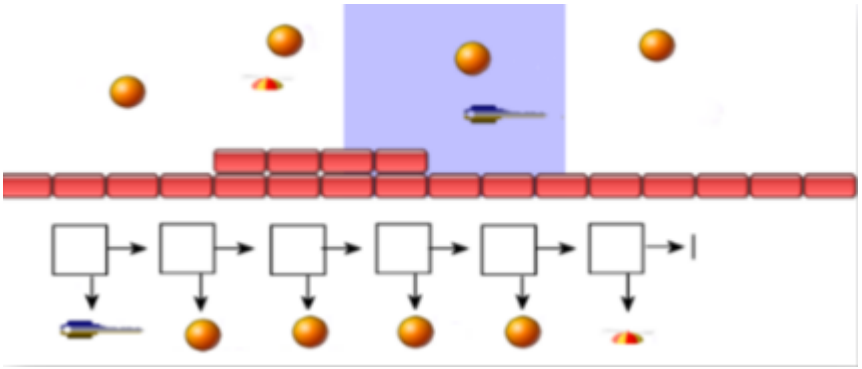
2. What is **sectioning**, and how does it work?
   - Section up the game area
   - Calculate which sections are currently visible
   - For each visible section, draw the sprites it contains
   - With this approach we must note when a sprite moves into a new section



3. How can **ordered/unordered lists** be used to draw sprites and objects to the screen?
    - Ordered List
        - Create an ordered list of sprites where the leftmost sprite is first, and the rightmost sprite is last
        - Scan list from the left looking for first visible sprite
        - Draw sprites from this one until the rightmost nonvisible sprite is reached
        - Requires constant movement of sprite references up and down the list



    - Unordered list
        - Maintain an unordered list of sprites
        - Scan through the entire list every frame and work out if the sprite should be displayed
        - Very simple, brute force approach
        - Very inefficient for games with many sprites but fine for only 10-20 sprites



4. Describe some methods we can use for **collision detection** between a sprite and the tilemap.
    - Bounding Box
        - Simple
        - Has a lot of white space however, so objects may collide even though they appear not to

- Bounding Circle
  - More complex
  - Less white space so more accurate collisions; can use multiple overlapping circles to get a greater degree of accuracy.

```
public boolean BoundingCircleCollision(Sprite one, Sprite two)
{
        int dx,dy,minimum;

        dx = one.x - two.x;
        dy = one.y - two.y;
        minimum = one.radius + two.radius;

        return (((dx * dx) + (dy * dy)) < (minimum * minimum));
}
```

5. How can we detect **collision direction**?
   - A simple method is to check a few points around the object and find out which is closest to the object that the collision occurred with (e.g. measure from each corner of the sprite)
     - Can use Pythagoras theorem to measure distance between points, then select the shortest
     - May also wish to do this for testing collisions of sprites with the corners of tiles on our tile map

6. Describe some issues with determining **when** and **where** two elements have collided.
   Because sprite positions are usually updated at discrete points, it is possible that a sprite could pass through or into an object between frames, those collision detection will mess up and the object may become stuck or some other problem will occur.
   Even more of an issue in multiplayer games because the machines will all run at different speeds.

   A solution is to compute a bounding shape using the before and after object positions to form the edges; this is accurate but computationally expensive.
   Otherwise, we could compute more frames than are actually displayed – if we had twice the number of animation frames, a collision would be detected in the new frame between two in the old method.

7. Describe the general **animation loop** of a 2D game.
   - Get user input
   - Update sprite positions in game world
   - Detect and handle collisions
   - Update display
     - Display background image
     - Display tile map
     - Display sprites
   - Repeat

SOUND

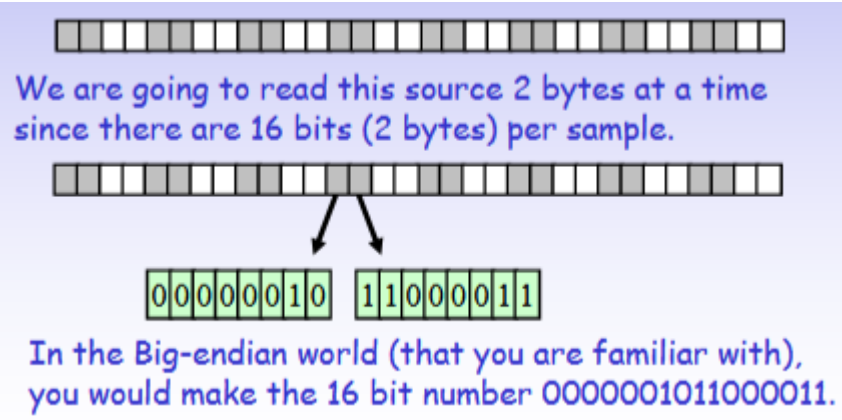
1. Describe the **5 main components** of playing a sound in Java.
    - File – a reference to the file to be played
    - AudioInputStream – the stream of sound data
    - AudioFormat – information about the format of the data (16/32bit, sample rate (e.g. 44,100Hz) mono or stereo)
    - Line(or subclass e.g. DataLine) – a way of connecting the AudioInputStream to the Java sound system
    - Clip – a way of controlling the playback of the sound

2. What is a **sound filter**?
   A sound filter can be applied to a sound source to alter the content of the source in some way. A common filter could be an echo, muffle or reverberation.

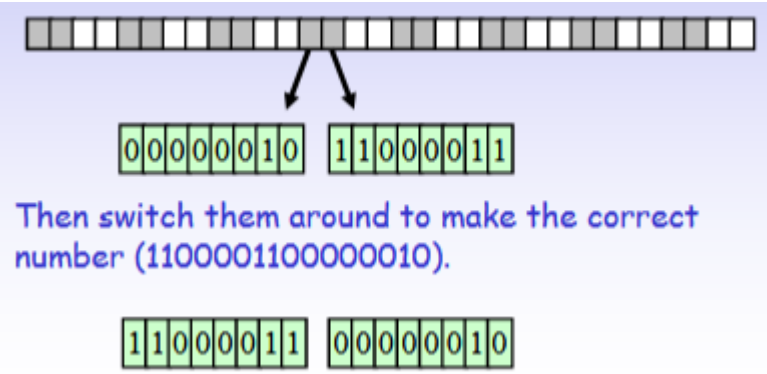
3. Give some examples of **sound filters**.
   Could have an echo, muffle, reverberation, and so on.

4. Write pseudocode for a **simple sound filter**.
   No pseudocode, but the general process is:
    - In Java, we can implement a sound filter by extending the FilteredSoundStream class
        - Add our own tricks to a new 'read' method
    - Also need a couple of methods to get and set the sound data since 'WAV' files are little-endian
        - Big endian data has the most significant byte first
        - What we are used to
        - Little endian data has the bytes switched round with the least significant byte first

5. What is meant by **big endian** and **little endian**?
    - Big endian – take a data stream of bytes coming from a sound source (e.g. WAV file) and read 2 bytes at a time (if 16 bits (2 bytes) per sample), then convert to 16-bit number



    - Little endian – read in order as before, and flip bytes around to get 16-bit number (used in WAV files)



6. Describe the process of using a **SoundFilter** class.
   Should create a SoundFilter class which is passed to a FilterInputStream to achieve the desired filter. The FadeFilterStream class shows the principle behind manipulating a sound signal.

7. Why should we play sounds in **separate threads**?
   If you do not use threads, then keyboard control would not be returned until the sound had finished playing; using threads means that keyboard input can run alongside multiple different sound sources at once, like sound effects.

8. Give some examples of **uncompressed**, **compressed**, **lossy compressed** and **music notation** formats of music.
   - Uncompressed
     - CD
     - WAV
   - Compressed
     - LA
     - FLAC
   - Lossy compressed
     - MP3
     - Ogg Vorbis
     - Real
   - Music notation
     - MIDI

9. How does **MIDI** work?
   - Music stored as a form of notation
     - Multiple tracks, for each track: instrument bank, note on, note off, volume
   - Very compact, files are a few K rather than MB
   - Reproduced sound dependent on quality of sound bank for the reproduced instrument
   - Allows a track recorded via a keyboard to be played back as a guitar
   - Only records music – not vocals

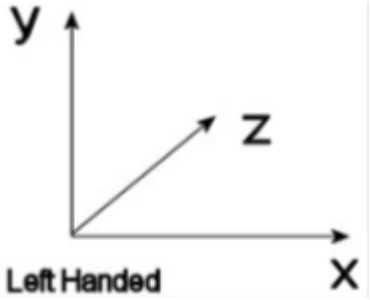
10. Why might we want to use MIDI? Why not?
    MIDI is good for adapting music in response to gameplay, i.e. it is easy to change if you go underwater or into a tunnel, create a sense of urgency, or something. It does not work for vocals and generally does not sound very good so you might want to avoid it.

3D

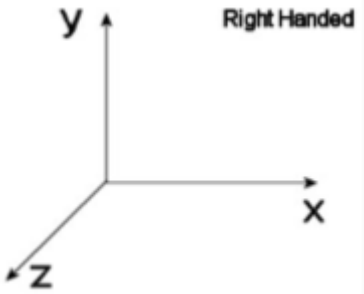
3D GRAPHICS OBJECTS

1. What **two main things** do we need to create 3D scenes?
   Perspective and shadows.

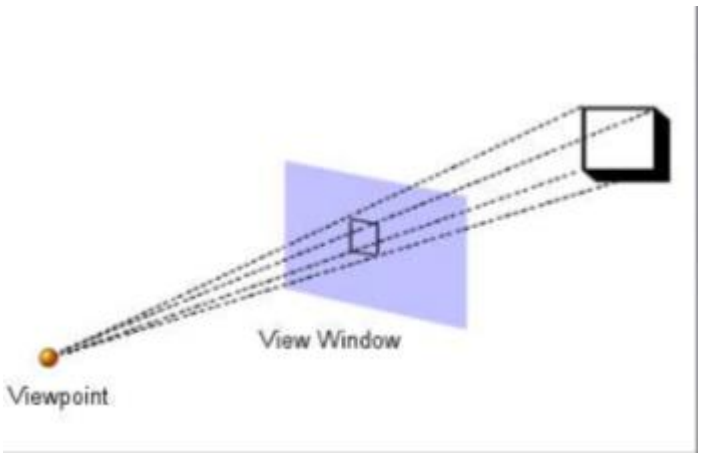
2. Describe the 2 main **coordinate systems** in 3D programming.
   - Left-handed – z increases as you move away from the viewer, positive rotation is clockwise about the axis of rotation



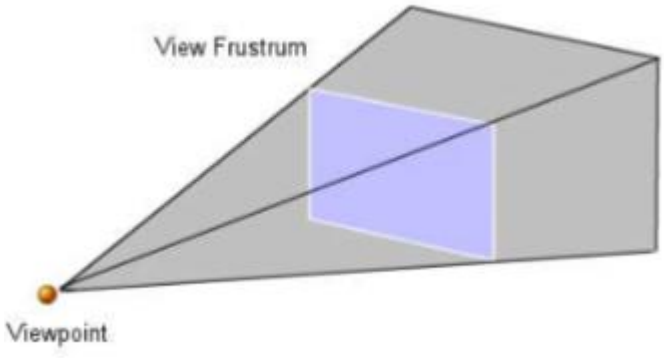
   - Right-handed – z increases as you move towards the viewer, positive rotation is anti-clockwise about axis of rotation



3. What is a **view window**?
   The 2d view window into the 3D world provided by the application displaying a 3D scene. As the user moves around, the 2D view changes.



4. What is the **view frustrum**?
   A 4 sided pyramid defining the visible area the user can see, which is broader or narrower depending on how close to the view window the observer view point is projected to be.



5. Describe the process of **projecting a point onto the screen** along with the **formula** for doing so.
    - Object is distance **z** along z-axis from viewer
    - And distance **x** along x-axis
    - View window is distance **d** from viewer
    - What is pixel **x'** along x-axis to the window?
    - Right angle triangle problem
    - **X' = (d/z) * x**

    General formula:

    - Establish a coordinate system and its origin (0,0,0)
    - Specify size and position of view window and viewpoint (establish view frustrum)
    - Draw object in 3D world coordinates e.g. drawCube(x,y,z,height,width,depth);
    - Project 3D object onto 2D array of screen pixels

3D GRAPHICS PROGRAMMING

1. What is the difference between the **2D** and **3D graphics pipeline**?
   A 2D graphics pipeline turns vector based 2D objects into pixel colours, whereas a 3D graphics pipeline turns 3D objects into screen pixels.

2. How is a 3D object typically **defined**?
   By vertices. Complex object approximated by flat, triangular surfaces defined by 3 vertices.

3. What components determine the **colour** of a vertex?
   Intrinsic colour plus lighting effects.

4. How are **non-vertex colours** determined?
   Through interpolation/shading model.

5. Name some common **3D graphics libraries**.
    - Direct X
    - OpenGL
    - Equivalent to Java2D in the 3D world
    - OpenGL graphics pipeline

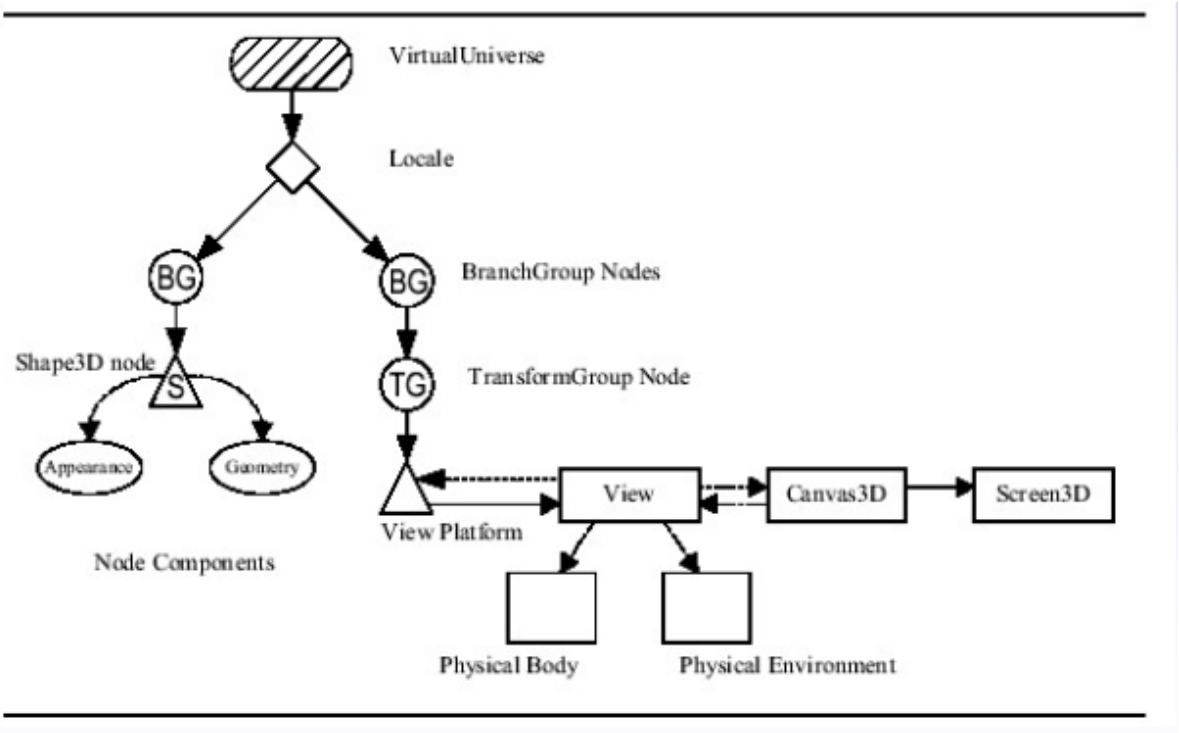
6. Compare the power of a typical GPU with a typical CPU in terms of **processing power and speed**.

NVIDIA GeForce GTX 1080 achieves 8800Gflops; 10-20 Gflops for current CPUs.

7. Name some of the **low**, **intermediate**, and **high-level** parts of 3D software.
    - Low level (OpenGL, DirectX) – Bindings: JOGL, WebGL
    - Intermediate (objects/scene graph) – Java3D, JMonkey, Three.js
    - High (3D IDE plus intermediate level) – Unity, Unreal Engine, Blender

8. What is a **scene graph**? What does it specify?
   A scene graph is a flow graph specifying elements of the 3D world, including visible objects, lighting, and cameras.

9. What part of Java3D handles the **low-level details** of drawing a 3D scene graph?
   The Java 3D renderer.
10. Draw an example of a **Java3D scene graph** (not including the view branch graph)



11. Understand all the **components** of said graph.
12. Name some **components of Java3D**.
    - A virtual 3D universe
    - Camera (or viewer) position in that universe
    - Lights
        - as many as needed
        - different locations and properties
    - Background
    - Objects in the 3D world
        - Scenery
        - Game sprites
        - Position and appearance
    - Objects can share properties
        - Appearance

- Transformations

13. What is an **alpha object**?
    An alpha object counts time, and for example can loop continuously with a period of 4 seconds.

14. What is the **Java3D rendering loop**?
    Intrinsic to Java3D. Renderer starts running in an infinite loop when an instance of View becomes live in the virtual universe, e.g. on creation of a SimpleUniverse. The renderer executes the following loop:

```
while(true) {
    Process input
    If (request to exit) break
        Perform Behaviors
        Traverse the scene graph
        and render visual objects
}
Cleanup and exit
```

JAVA 3D SCENE GRAPH

Some of these questions overlap slightly, but I feel it is important to practice.

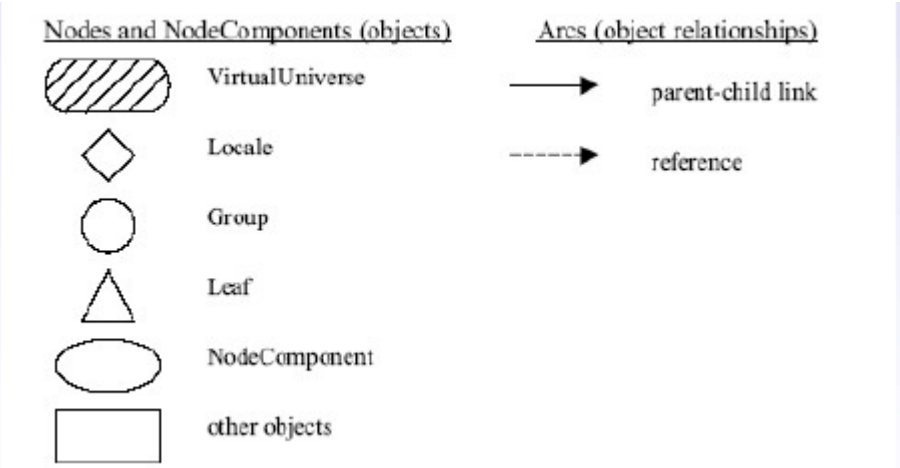
1. What is a **scene graph**? What does it specify?
   Describes the components of a Java 3D application, using a tree structure.

2. Describe the **structure** of a Java3D scene graph.
   - Created using instances of Java 3D classes
     - Defines geometry, sound, lights, location, orientation and appearance of visual and audio objects
   - An arc may also describe a reference relationship between a node component and a node
     - Node components describe appearance attributes
     - Node components and reference arcs are not part of the scene graph tree
   - A single path exists between a root node and a leaf
     - Scene graph path of the leaf node
     - Defines the state of the leaf node

3. What are the 2 main components of the **tree structure**?
   Nodes and arcs.

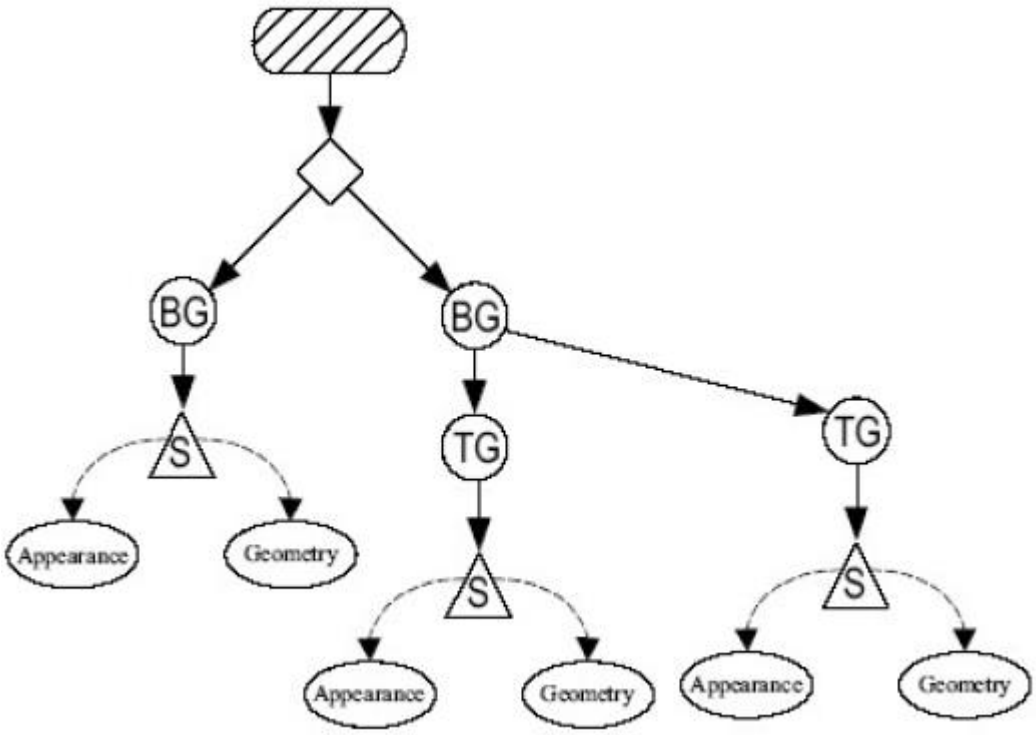
4. Describe some of the **rules** for an arc.
   Arcs commonly describe a parent-child relationship
   - All nodes have a single parent
   - Group nodes can have many children
   - Leaf nodes have no children

5. Draw and describe each **component** of a scene graph.
   - A scene graph defines the virtual universe
   - A scene graph tree is rooted at a locale node



- Locale node provides a reference point for location of all objects in the virtual universe
  - Landmark
  - A virtual universe may have more than one locale
- Group nodes group components together
  - Branch groups
  - Transform groups
- Branch groups are roots of subgraphs
  - Content branch graph
  - View branch graph
- Leaf nodes are visual or audible components
  - Shape3D
  - behaviours

6. What is featured in the **content branch graph**?
   Most programming work is devoted to the content branch graph; includes branch nodes, transform nodes, leaf nodes

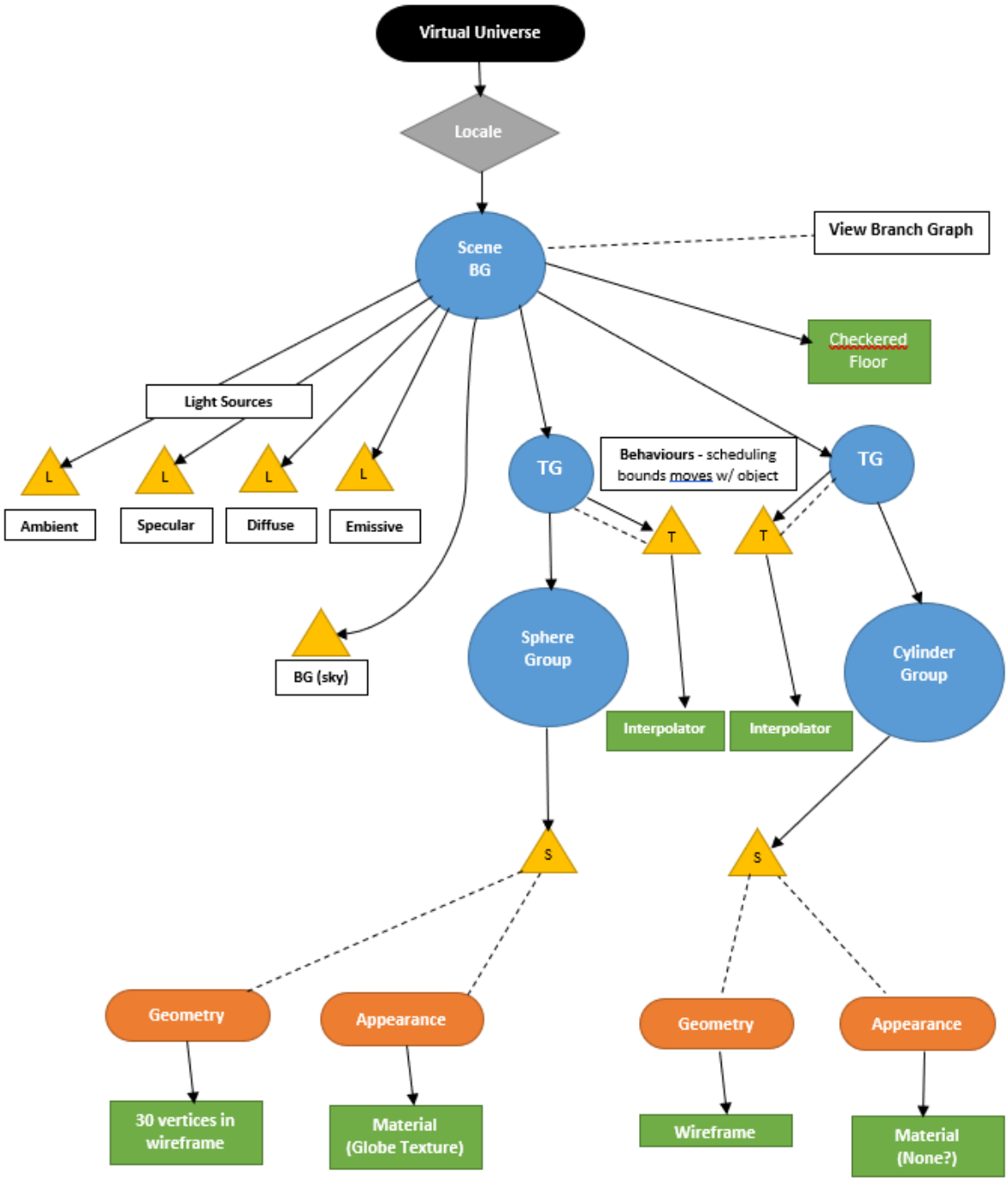


7. Describe some **illegal features** of a scene graph that you could have.
    - Scene graphs may not contain cycles
    - Multiple parent runtime error

8. How does **compiling** a scene graph work?
    - Creates internal representation of scene graph
    - Can perform optimisations

9. What is a **capability**?
   Capabilities specify components that may change following compilation.

10. Draw an example of a scene graph for the **sphere practical**.



INTERACTION & ANIMATION IN JAVA 3D

1. What is the difference between an **interaction** and an **animation**?
    - Interaction is a change in the scene in response to user action
        - Key press
        - Mouse click or movement
    - Animation is a change without any direct user input
        - Passage of time
        - Collisions

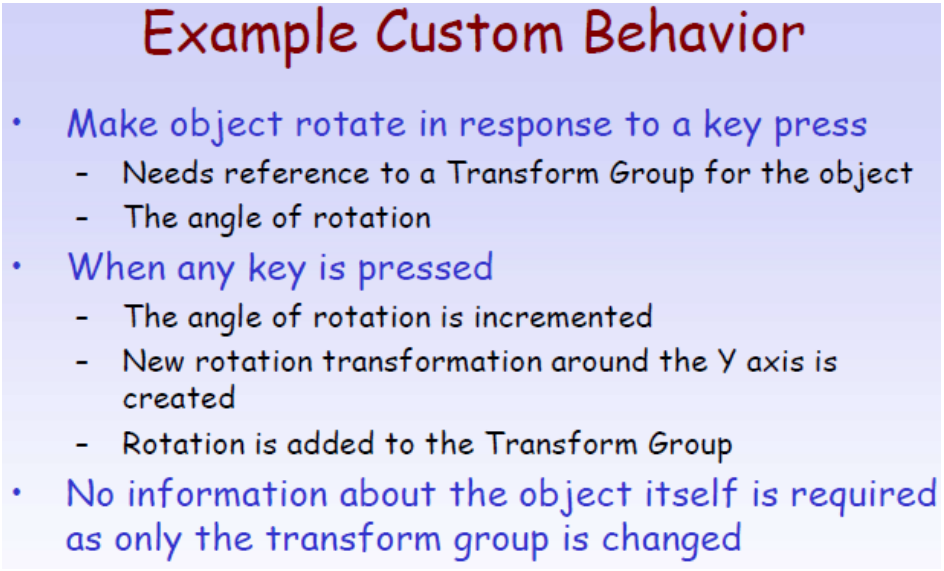
2. Do both **interaction** and **animation** cause changes to the scene graph?

Yes.

3. What is a **behaviour**?
   A link between a stimulus and an action.

4. Describe some **inbuilt behaviours**.
   - KeyNavigatorBehaviour
   - MouseBehaviour – includes MouseRotate, MouseTranslate and MouseZoom
   - PickMouseBehaviour – includes PickRotateBehaviour, PickTranslateBehaviour and PickZoomBehaviour

5. Describe the process of writing a **behaviour class**.
   - Behaviour is an abstract class
   - Custom Behaviour class implements
     - The class constructor
     - initalize() method
     - processStimulus() method
   - Behaviour typically has an object of change
     - e.g. visible object; view platform
     - a reference to the object of change is usually set up in the Behaviour constructor
   - Initialize() method
     - Executed when behaviour first goes "live" (ie when added to scene graph)
     - Sets initial trigger event for the behaviour
     - Trigger is a (combination of) WakeupCondition object
     - Also sets initial values of any behaviour state variables
   - processStimulus() method
     - invoked when the trigger event occurs (e.g. key press; passage of time)
     - carries out action in response to the event
       - manipulates object of change
       - usually results in a change to the scene graph
     - resets the trigger (if necessary)

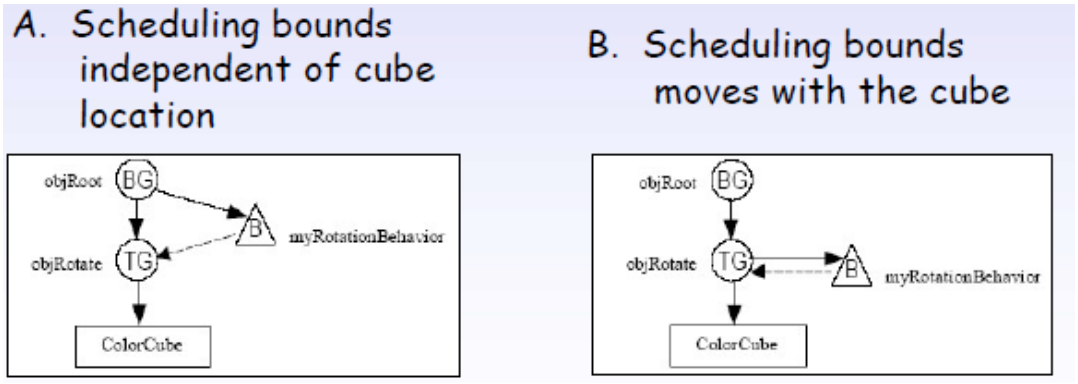


6. How can we use a **behaviour**?

- Add transform group to scene graph above the object of change
- Insert behaviour object into scene graph, with reference to the object of change
  - Via the transform group
- Specify the scheduling bounds for the behaviour
  - Behaviour only active when current view intersects the scheduling bounds
  - Behaviour is never active if bounds not specified
- Set write (and maybe read) capabilities on the transform group
  - So, it can be changed at run time

7. How do we **add a behaviour** to a scene graph?
   - Behaviour can go anywhere in the scene graph
   - Possible considerations are:
     - Code maintenance
     - Effect on scheduling bounds

8. Draw an example of how a behaviour could be **represented** in a scene graph.



9. Describe some examples of **wakeup conditions**.
   - WakeupOnAWTEvent – triggered when an AWT event occurs, e.g. key pressed, key released, mouse clicked, mouse dragged
   - WakeupOnElapsedTime – wakeup when a specific number of milliseconds has elapsed
   - WakeupOnElapsedFrames – wakeup when a number of frames have elapsed
   - WakeupOnCollisionEntry
   - WakeupOnCollisionExit

10. Describe some examples of **behaviour utility classes**.
    - KeyNavigatorBehaviour – provides precoded object movement actions in response to certain key presses; pass it the transform group of the object to be controlled
    - Mouse interaction – translating, zooming and rotating visual objects; again, provide transform group of visual object to behaviour
    - Picking – selecting a visual object with a mouse click using a "pick ray"

ANIMATION IN JAVA 3D

1. What is **time-based animation**?
   Animation is a change without any direct user action, time-based depends on time, and uses interpolators, alpha objects, and custom behaviours.

2. What is **collision detection**?

Same as for 2D, uses bounding boxes and spheres.

3. How do we use an **alpha object**?
   A value from 0.0 to 1.0, which changes as a continuous function of time.
   - Synchronized against Java 3D system start time
   - Four phases
   - Patterns using one to all four phases
   - Fixed number of cycles or continuous
   - Waveforms can be smoothed – useful for acceleration/deceleration effects

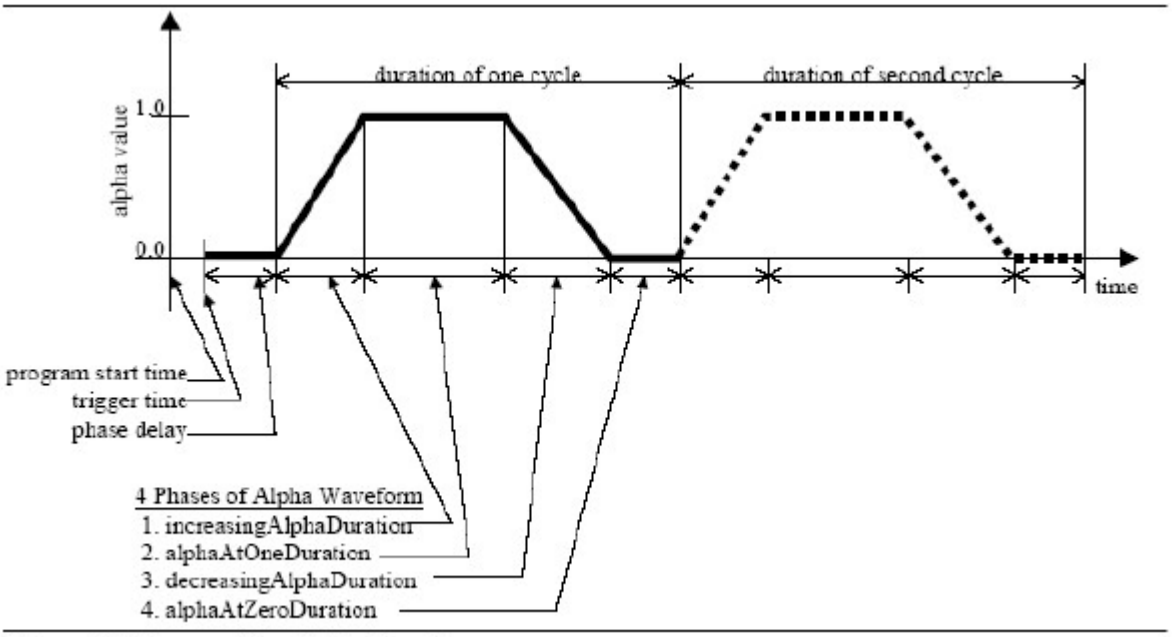


Figure 5-2 Phases of the Alpha Waveform.

4. Describe how an **interpolator** works.
   - Change a property of a visual object
     - Location (translation) and orientation (rotation)
     - Size, colour and transparency
   - Property changes over time
     - Interpolated against an Alpha object value
   - Built-in interpolators
     - Utility classes provided for interpolating most object properties of interest

5. What is **hand-crafted animation**?
   - Can write custom behaviours that use Alphas
     - value() method of Alpha object
   - But movement in games is not always regular
     - Determined by interaction between sprites
     - Game state
   - Custom animation behaviours using time or frame-based wakeup criteria
     - Identical to 2D game approach

6. How does **collision detection** work in Java 3D?
   - Collision detection based on bounding regions of objects
     - Boxes & spheres

    - More complex polytopes
    - Define explicitly for each object of interest
- Need to detect when bounding regions of colliding objects overlap
    - Bounding regions in Java 3D have a built-in method intersect() that can detect overlap with another bounding region
- Take appropriate action in response to collision
    - E.g. rebound

7. Describe the process of **implementing collision detection** in Java 3D.
    - Give each object an associated bounding region
        - Cube, sphere
        - E.g. BoundingSphere playerbnds = new BoundingSphere();
        - When an object moves, its bounding region must be moved with it
            - Manually apply transform or recreate bounds at new location
            - Add bounds to a BoundingLeaf node
    - Time-based behaviour checks for (predicted) intersections between bounds of visible objects
        - Schemes for checking similar to 2D
        - E.g. if (playerbnds.intersect(enemybnds)) {fix collision}

ADVANCED ANIMATION

1. What is **billboarding**?
    - Use of 2D images to represent 3D objects
        - Computationally cheap replacement for full 3D models
        - Typically used for complex real-world objects that are in the middle to far distance in the 3D world e.g. trees

2. How does it work, and how do we use it in Java 3D?
    - Images are continually rotated as viewer position changes so that they face the viewer
        - Look 3D as always viewed in a direction parallel to the surface normal
        - Ideal for cylindrically or spherically symmetric objects such as a water tower or the moon
    - Also used to keep textual information readable by viewer from any angle

3. What is the default **rotation** for billboarding? What other ones can we have?
   Default is around an axis, but can also rotate around a point which is good for spherical objects like the moon.

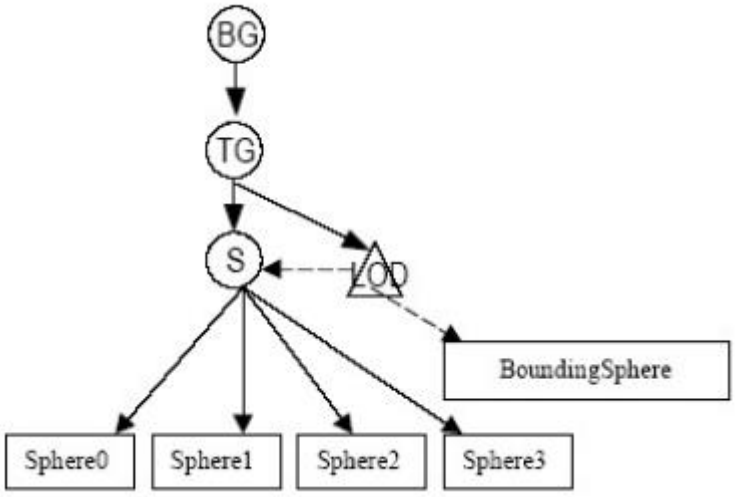
4. Describe **Level of Detail** (LOD).
   Variation of the amount of detail in a visual object depending on its distance to the viewer. A close object would have good geometrical and appearance detail for example.
    - LOD may also be changed in other circumstances
        - Based on rendering speed
        - Based on object velocity
        - Could also be user settable to improve performance

5. How is LOD **implemented** in Java 3D?
    - LOD object
        - Abstract class
        - DistanceLOD extends LOD to provide “switch on distance to viewer”

    - Custom LOD classes can be written
- LOD references one or more Switch objects
- Switch objects allow switching between different visual objects
    - A Switch object is a special group that allows zero, one or more of its children (usually visual objects) to be rendered
- Switching depends on specified criteria
    - E.g. distance to viewer

6. Draw a scene graph of a typical **DistanceLOD**.

- Create a number of visual objects with varying detail
- Add objects to a Switch object
- Set Switch object as target of DistanceLOD object
- Selection of child of Switch object to be rendered depends on a set of threshold distances
    - Distances specified as an array starting with the maximum distance at which the first child will be used
    - Switch to second child made when this distance exceeded
    - First child is most detailed visual object ie one to be used when close to viewer
    - Subsequent distances must always be greater than previous distance
    - One fewer distances need to be specified than there are visual objects to be rendered



7. Describe a **switch object** and its uses.
   Controls which of its children should be rendered (zero, one or more). Children are sub scene graph branches.

8. How is it **implemented**?
    - Children added using addChild() method
    - Children selected for rendering using setWhichChild(int Child) method
        - Numerical index depends on order children added to Switch
        - Switch.CHILD_ALL and CHILD_NONE to show and hide all children respectively

9. What is **picking**?

- User places mouse pointer over visual object and presses mouse button
- Ray projected into the visual world from the mouse pointer position
- Closest object to image plate that intersects the ray is "picked"

10. Describe some **possible uses** of picking.
    Could move, rotate, change appearance or geometry, make invisible, etc.

11. What should be **returned** by the picking animation?
    The scene graph path to the visual object.

12. Describe some **features** of the pick utility class.
    Utility behaviour classes provided for object rotation, translation and zooming

3D OBJECT MODELLING

1. What is a **polygon**?
   A closed planar (flat) figure with 3 or more sides.

2. What is a **polyhedron**?
   3D shape built from flat 2D polygons.

3. Why should polygons and polyhedrons be **convex** instead of **concave**?
   Concave shapes are not easy to fill quickly so it is inefficient.

4. How many **sides** can a polygon have at max in Java 3D?
   Java 3D will only render polygons with at most 4 sides; complex polygons must be subdivided, usually into triangles.

5. Describe the process of **hidden square removal**.
   - Situation where we have two polyhedrons, one on top of the other
   - We would like to keep the number of polygons we draw to the minimum necessary
     - All the back-facing polygons are hidden
     - One of the forward-facing polygons in the grey cube is hidden
     - There are 12 (2x6) polygons we could draw and fill but only 5 are actually visible
     - Need to draw them in the right order...

6. Describe the process of **back face culling**.
   - In this approach we only draw the polygons in an object that we know are facing the camera
     - Each polygon has a 'normal' – a vector that is at right angles to the face of the polygon
     - If angle between the camera and the normal is less than 90 degrees, the polygon is facing the camera
     - All the polygons on the back of the above cube have a normal that faces away from the camera i.e. their normal is greater than 90 degrees
     - We can calculate this angle using the 'dot product' of two angles

7. Describe the **painter's** and **reverse painter's algorithm**.
   - Painters' algorithm
     - Draw all the polygons from the furthest away through to the nearest
     - Nearest polygons draw over the furthest

- Somewhat inefficient since we overdraw so many times
- Reverse painters' algorithm
  - Draw nearest polygons first
  - Subsequent draw commands which try to fill a pixel that has already been drawn are ignored since we must now be behind a front facing polygon

8. What is **z-buffering**?
   - A buffer is created the same size as the screen (called the Z-buffer because it stores information about the Z or depth access).
   - Each pixel in a polygon is assigned a depth based on its z coordinate
   - An attempt is then made to draw each pixel in the polygon onto the Z-buffer
   - If the value in the Z buffer is greater than the one allocated to the pixel in the polygon, the pixel is drawn, and its value becomes the new value in the Z buffer
   - This allows slanting and overlapping polygons to drawn correctly

JAVA 3D OBJECT MODELLING – GEOMETRY

1. Describe the 3 main ways to create **physical objects** in Java 3D.
   - Basic geometric primitives
     - ColorCube
     - Box, Cone, Cylinder, Sphere
   - Vertex specification via geometry classes
     - TriangleArray, QuadArray
   - Loading external models
     - Model created via specialist software, e.g. Blender
     - Loading software to convert geometry specification to Java 3D format

2. What is the **purpose** of the Shape3D class?
   Instance of Shape3D defines a visual object.

3. Draw a scene graph showing the **typical components of a Shape3D object**.

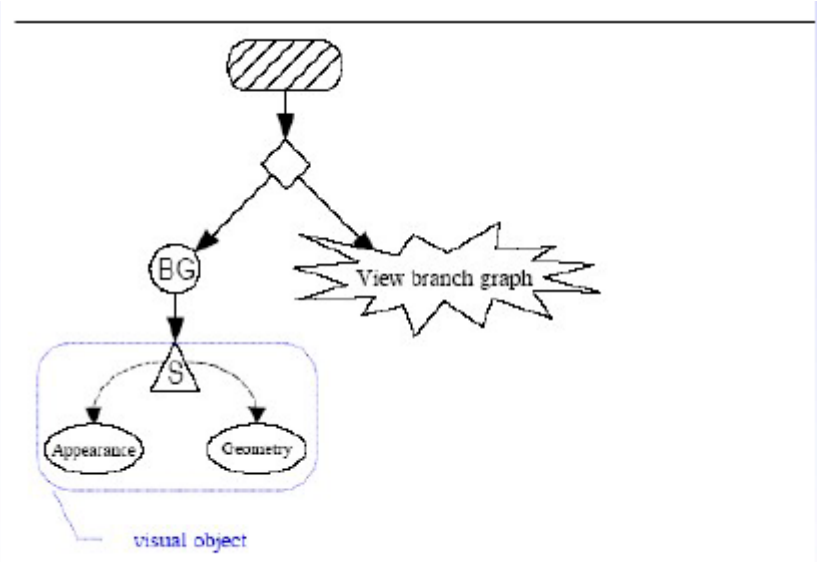


Figure 2-2 A Shape3D Object Defines a Visual Object in a Scene Graph.

4. What is a **primitive visual object**? Give examples.

- Inbuilt basic geometric shapes
    - Box, Cone, Cylinder, Sphere
- Extend the Primitive class
- Contain more than one Shape3D

5. What is a **mathematical class**? Give examples.
    - Several mathematical classes are provided to make geometry creation possible
    - Specify vertex-related data
        - Point – for 3D coordinates
        - Colour – for colours
        - Vector – for surface normals
        - TexCoord – for texture coordinates

6. What is a **geometry class**? Give examples.
    - Use subclasses of GeometryArray class
    - Specify sets of vertex coordinates
        - Plus, appearance-related data, such as vertex colour

JAVA 3D OBJECT MODELLING – APPEARANCE

1. What methods can we use to **specify vertex colours** in Java3D?
    - Geometry node component
    - Appearance node component
    - Material node component
        - Final colours determined by scene lighting

    Non-vertex pixel colours determined by a shading model.

2. What is an **appearance bundle**?
    Appearance node component referencing various appearance attribute node components.
    An appearance node component does not contain appearance information itself, but references other appearance attribute node components that specify how the visual object will look.

3. Name some examples of **appearance attribute node components**.
    - PointAttributes
    - LineAttributes
    - PolygonAttributes
    - ColoringAttributes
    - TransparencyAttributes
    - RenderingAttributes
    - Material
    - TextureAttributes
    - Texture
    - TexCoordGeneration

4. Can multiple appearance objects **share** attribute components?
    Yes, this may improve rendering performance.

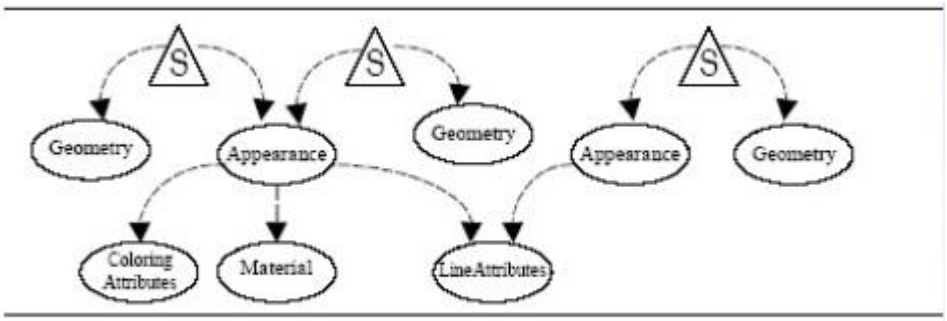


Figure 2-21 Multiple Appearance Objects Sharing a Node Component

5. Describe some examples of **attribute classes**.
    - Point attributes
        - Vertex rendered as a single pixel by default
        - Can increase size of square "point"
        - Specify antialiasing to make points look more rounded
    - Line attributes
        - Solid, one pixel wide, not antialiased by default
        - Can specify style, width and antialiasing
        - E.g. dash, dot, dash-dot
    - Polygon attributes
        - Drawn filled with back-face culling by default
        - Can specify wireframe rendering
        - Can specify front-face or no culling
    - Colouring attributes
        - Specify vertex colours
        - Specify how pixels between vertices are coloured using interpolation of vertex colours
            - Flat or Gouraud shading
        - Colours specified in Geometry node component will override an Appearance colouring attribute
        - Appearance colouring attribute also ignored if scene is lit
            - Colours determined by Material plus lights
    - Transparency attributes
        - Specifies transparency of the visible object via an alpha level
    - Rendering attributes
        - Controls per pixel rendering
        - Depth test: determines whether depth buffer is used for hidden surface removal
            - Z-buffering
        - Alpha test: determines whether alpha is used to give transparency

6. What is **face culling**?
    - Front face of a polygon is determined by vertex ordering
        - Counter clockwise

- (Right-hand rule)

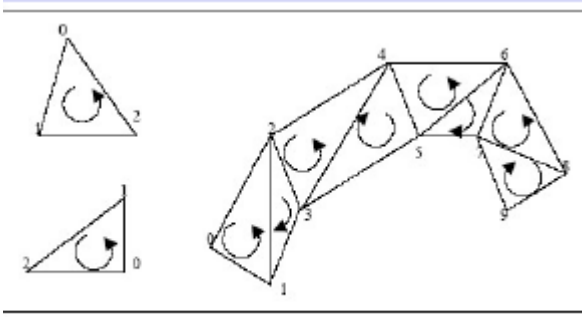


Figure 2-24 Determining the Front Face of Polygons and Strips

7. What is the difference between **back-face, no-face** and **front-face** culling?
    - Back-face culling is the default setting
    - No-face or front-face culling may sometimes be more appropriate
        - E.g. checkered floor
        - Mobius strip

LIGHTING

1. Name the components of a Java3D **lighting model**.
    - Surface normal
    - Light direction
    - Viewer direction

2. What **3** types of **light reflection** can we get from an **object**?
    - Ambient - results from low level background light
    - Diffuse – normal reflection from a light source
    - Specular – highlight reflections

3. How does a **shading model** work?
    - Lighting model first shades each vertex
        - – Determines colour based on each light source
    - Remaining pixel colours determined from vertices
        - – Flat or Gouraud shading

4. What is the difference between **flat shading** and G**ouraud shading**?
    - Flat shading chooses colour of one vertex for all pixels in enclosed polygon
    - Gouraud shading uses trilinear interpolation of colours from all vertices of a polygon

5. Describe the **4 main types** of lights.
    - Ambient
        - Light of same intensity in all directions and at all places
        - No location
        - Colour
        - Results in flat shading of lit objects
        - Used in combination with other light source types
    - Directional
        - Light from one direction only (e.g. the sun)
        - Light vector (L) is constant

    - Direction and colour but no location
- Point
    - Omni directional light whose intensity decreases with distance from source (location)
    - Location and colour but no direction
- Spot
    - Subclass of point light with direction and concentration
    - Only light capable of lighting only a portion of an object

6. What types of lighting can we get from **materials**?
    - Material specifies colours and/or texture
    - **Ambient, diffusive** – colour of object
    - **Emissive** – glow-in-the-dark
    - **Specular, shininess** – reflective highlights

7. What is the **Phong Lighting Equation**?
    - Combines dull diffuse base with specular highlights
    - Approximates the look of plastic
    - C=Cd.Cl(N.L) + Cs.Cl(R.E)s
        - R is the reflection vector (from N and L)
    - Contributions from individual lights are summed

8. Describe the process of **constructing** a lit scene.
    - Construct lights
        - Typically, an ambient light plus 1 or 2 directional lights
        - Set influencing bounds for each light
        - Add lights to scene graph
    - Construct visible objects
        - Specify surface normals
        - Add light-enabled material
        - Add objects to scene graph
    - If any of these steps is missed out, then an object will not be lit
        - E.g. easy to forget the influencing bounds for a light, or even to add the light to the scene graph

9. What is the **region of influence** of a light? How is it **determined**?
   The region that is affected by the light (relative to light's location for point and spot lights), determined by its influencing bounds.

10. What things can we do to **more closely approximate realistic lighting**?
    - Light scoping
        - Only enable lighting on objects that should be lit
    - Fake shadows

TEXTURES

1. Why can't we go into **complex detail** with textures, such as bark on a tree or leaves?

Way too computationally expensive, too many polygons, details, etc. Can still achieve good visual appeal by using a 2Dimage as a surface texture on a simple geometry.

2. What is **texture mapping**?
   Fitting a texture onto a polygon rather than just using a plain colour, in order to make it look more realistic.
   - Basic problem is how to map the 2D image to the surface of the polygon
     - Images and polygons may be different sizes and shapes
   - Textures used to fill a polygon are often smaller than the area they are filling and need to be repeated
     - Choosing a suitable isomorphic pattern can make this less obvious
     - Same principle as background patterns on PC screens or web pages

3. What is a **texel**?
   A texture element – a pixel in a texture.

4. What is a **MIP map**?
   Applying texture images of smaller size as the visual object moves further away from the viewer (**m**ultum **i**n **p**arvo – many things in a small place)

5. Describe the general process for using **textures** in Java3D.
   - Prepare image files: each dimension must be a power of 2 for efficiency in texture mapping e.g. 256 x 128
   - Load image into a Texture object
   - Add texture to an Appearance bundle
   - Specify texture coordinates for object geometry

6. Why can it be easier to map to a **sphere** or **cylinder** object than a **made-up polyhedron**?
   Primitive objects like a sphere can specify texture coordinates automatically.

7. How can we set up **texture coordinates**?
   - Need to relate the texture image to the surface of the object to be textured
   - Set up mapping between texture coordinates and the vertices of the object geometry
     - Texture coordinates are (s,t)
     - Vertex coordinates are (x, y, z)
   - E.g. map a square image onto a Quad array (square)

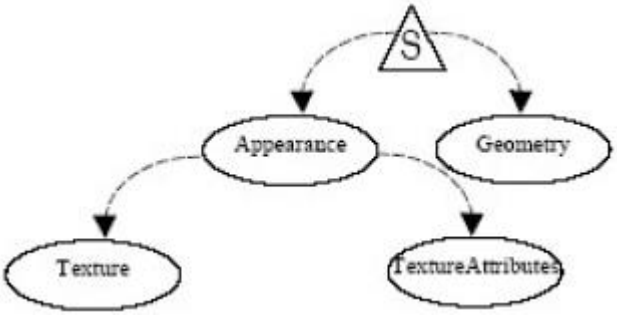
8. How can we map **non-vertex coordinates**? How are texels for **vertices** and **non-vertex** pixels determined?
   - Rendering the textured visual object involves choosing a texel for each pixel corresponding to the object
   - Texels for vertices are defined by the texture coordinates
   - Texels for non-vertex pixels determined by trilinear interpolation

9. What choices can we make to **speed up** the process when **texture rendering**?
   Can use tiling of an image to provide a texel and can also use the colour at the edge of an image to save on processing.

10. What determines **final pixel colour**?

Depends on both texel colour and geometry colours & lighting.

11. Draw a section of a scene graph showing the **appearance bundle with texture and TextureAttributes components**.



SIDE NOTES

- **Bit depth** - 24 bit(8r, 8g, 8b, RGB) 32 bit(8r, 8g, 8b, 8a, RGB)
- **Screen res** - resX * resY * bit depth = 5120kb per frame
- **Opaque** - No transparency
- **Transparent** - 1 or more pixels are see through I.e. BG white colour
- **Translucent** - Each pixel has a defined level of which a background will show through
- **Anti-Aliasing** - Smooths jagged edges with blurring.