Sprites

Sandro Spina
Computer Graphics and Simulation Group

Computer Science Department
University of Malta

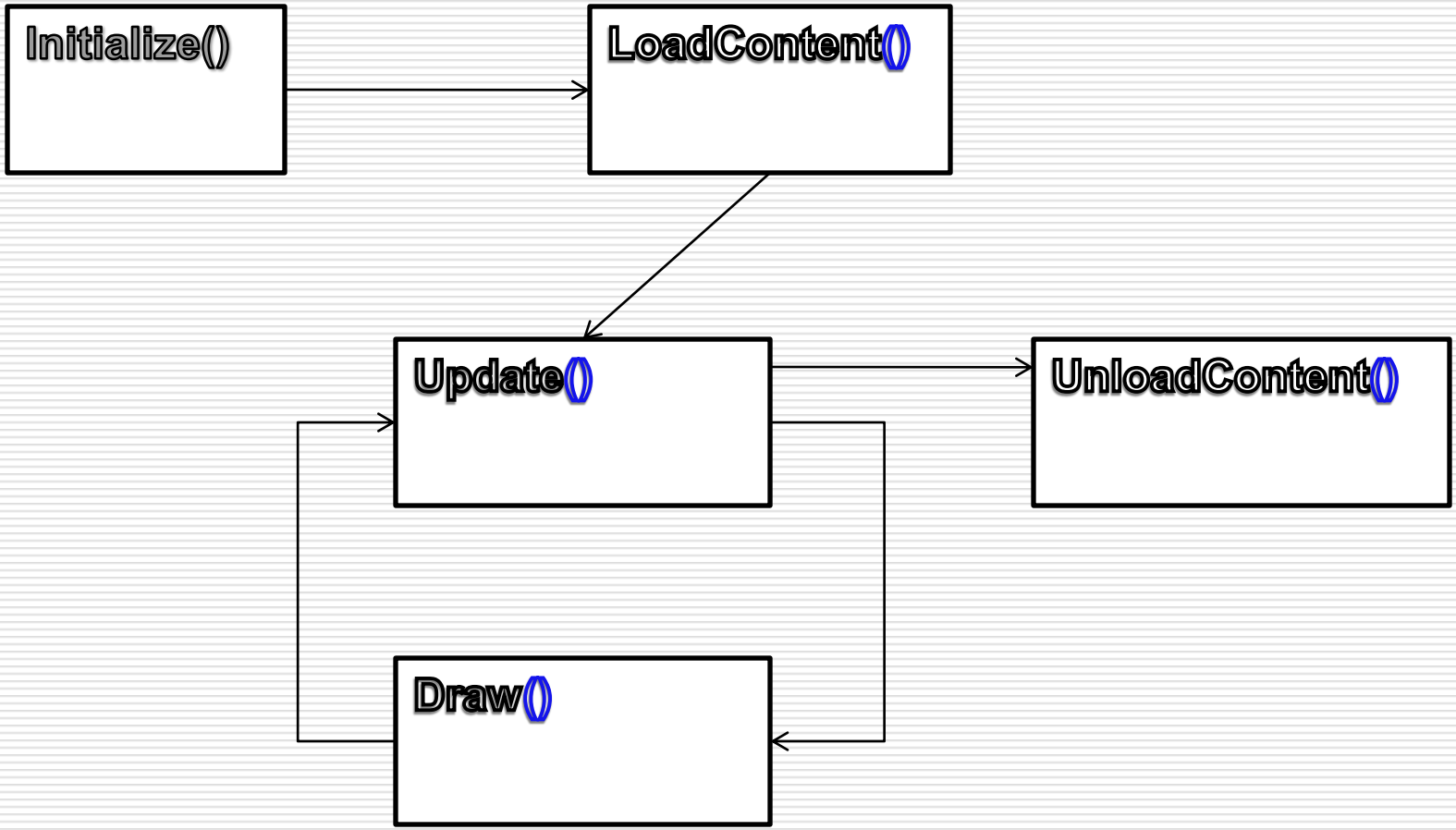
2D Graphics

- In 3D graphics one usually create a scene (in 3D space) then somehow generates (projects) a 2D representation of it on the screen (viewport)
- However if we do not need to represent anything in 3D we can directly use 2D (flat) objects ... such as sprites.
- It would be like playing a board game (or tablesoccer, subbuteo, whatever ...) on the surface of a table and having a camera directly above it pointing in a perpendicular direction towards the table.

XNA – the game loop

- Initialize()
  - Variables etc.,
- LoadContent()
  - Loading of images, sprites, models, sound, etc
- Update()
  - Game logic, send data over network, etc ...
- Draw()
  - Of course you draw your stuff here ....

XNA – the game loop



Sprites

- A sprite is essentially an image file (png, jpg, bmp, etc.), which ideally has a transparent background.
- It is represented as a 2D texture within XNA.
- Its properties are:
  - Position :: Vector2D
  - Rectangle in source image :: Rectangle

- Check Example ...

Content Pipeline

- The XNA framework helps you out in loading different content into your game such as Textures and Fonts.
- You'll find it in the Solution Explorer of VS Express.
- Content.Load<Texture2D>("spaceship");
- Use Transparent images (png)

Sprite Rotation ...

- Let's assume that we're creating a 2D asteroids game. You'll need rotation in order to be able to have your spaceship fire in all possible directions. Remember we are in outer space so our aircraft can rotate 360deg to shoot the asteroids.

- Sprites rotate around their origin ... but

- 'Problem' is that by default 0,0 in the local coordinate system of the sprites is the top left corner of the sprite.

- Solution (trivial) is that when we rotate we need to move the origin to the centre of the sprite. In XNA you need to set the pivot when creating the sprite.

Sprite Sheets

- Used to add animations to sprites ...
- ...using essentially the same techniques of a cartoon flipbook
- Check Prince of Persia example ...
- Check Three Rings example ...

- Check the code in XNA on how to loop through the 2Dtexture.

SpriteBatch Properties (i)

- You'll probably have one SpriteBatch object for your game in which you will draw all your 2D images.

- SpriteBlendMode
  - None: No blending of colours
  - AlphaBlend: Alpha values are blended. This is the default which enables transparencies.
  - Additive: sprite colours are blended in to the background colours.

SpriteBatch Properties (ii)

- SpriteSortMode (Defines the sorting options of rendered sprites)
  - *Deferred*: sprites are not drawn until SpriteBatch.End() is called.
  - *Immediate*: the Begin call immediately sets the graphics device settings and new Draw calls are immediately drawn.
  - *Texture*: same as Deferred, but sprites are sorted by texture before being drawn.
  - *BackToFront*: same as Deferred, but sprites are ordered in front-to-back order based on layer depth
  - *FrontToBack*: inverse of the above mode (Check example with the game background)

SpriteBatch Properties (iii)

- Another useful function within SpriteBatch is *DrawString()* which takes as parameters the string you want to display, the font, the coordinates where to display)

  - You can use this function to display scores. Or when someone is hit the information is displayed on all the clients (for eg. In multiplayer mode).

  - You can also use it to display messages between clients.

Moving Around ...

- We've seen that sprites have a location (Vector2D) which determines where on the window frame they are drawn.
- In order to create movement we trivially need to update (in the update() method) with every frame drawn.
- Let's consider 'movement' scenarios in the game of asteroids ....
- We could have either:
  - Automatic: Asteroids move on their own
  - User Controlled: You control you spaceship

… And following a vector

- Conceptually one can think of movement in a straight line simply as the task of following a vector's direction.
- Location (0,0) is at the top left corner of the window frame.
- The Vectors (1,0) and (0,1) define what?
- If you take them as positions (1,0) this only means one pixel to the right from (0,0) ...
-
- Speed can be encoded as simply the amount of pixels covered per frame (pixels/sec instead of metres/second)

User Input ... For Asteroids

- Arrow Keys can be used to control the direction (rotation) of the spaceship (sprite) ....
- Up key can be used to thrust forward
- ... space to fire the missiles !!!
- Check example code

User Input ... Acceleration !

- In outer space a space ship has thrust .... but no brakes!!!
- The implication of this is that the ship does not stop immediately but depending on the value of its thrust
- This can be encoded (in a simplified way) by thinking of thrust as increasing acceleration ...

User Input ... Acceleration !

- We can rotate our space ship without thrusting forward.
- This is very useful when you are trying to evade your enemy and at the same time trying to fire in his direction.
- Therefore .... Direction = last direction in which thrust was applied. Rotation alone does not change the direction of where you spaceship is heading.

Checking the Frame Rate

- Irrespective of whether you are using a GTX280 card or a meagre ATI X1270 (my laptop!!) card the experience (in terms of frame rates) should 'ideally' be similar. Especially true with network games.

- By default XNA runs at 60 FPS ... However you might want to animate you sprite sheet at 30FPS.

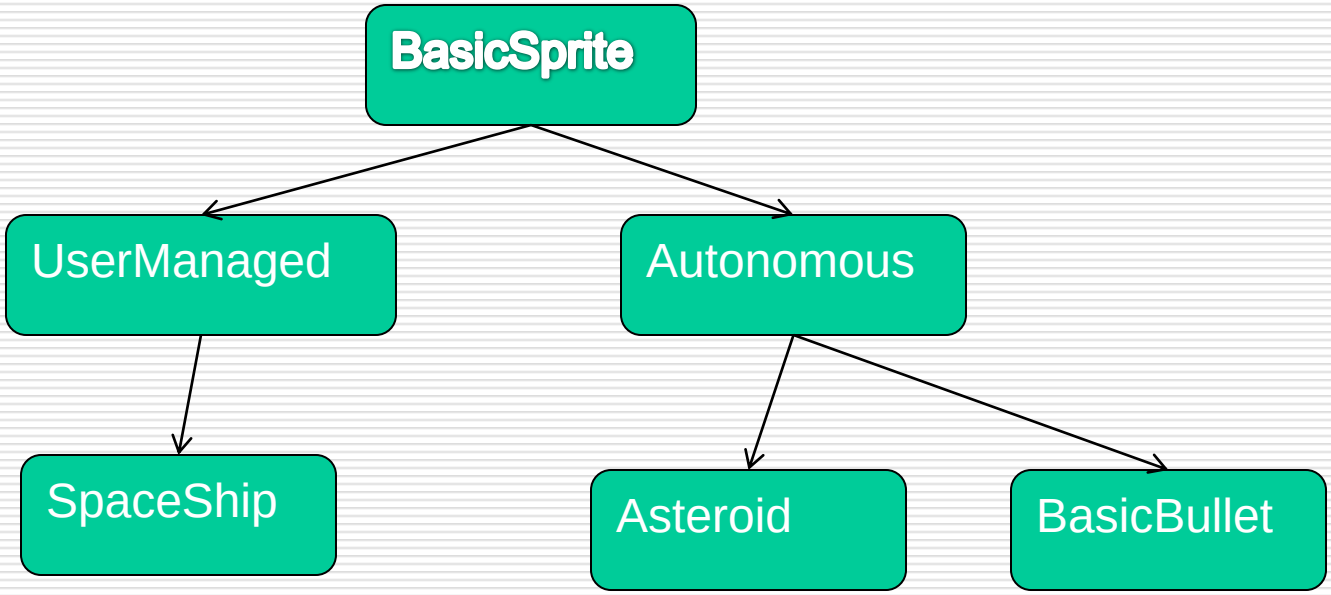
- You can fix the frame rate (animation speed perhaps is a better word) by using the GameTime parameter which is passed between each Update() and Draw() call.

- Also if you think your graphics card is not up to scratch you can check that out by checking the Boolean property IsRunningSlowly of the GameTime object.

Many Sprites = an OO design

- In any programming that you do (including games programming) you need to structure you code !!!!
- In the case of the Asteroids game ... you need to cater for your ship + opponent ships (amongst other things)
- Asteroids and bullets !!!!
- Therefore .... You need a good OO design in order to get everything running smoothly.
- You basically need to create a Sprite Hierarchy + a Sprite Manager

OO Design with a Sprite Manager



- A Sprite Manager is used to load content, call updates and draws of the different sprites.

XNA Help ... GameComponent class

- The XNA framework provides you with a nice feature that will help you out design your game.
- The GameComponent class essentially is used to integrate different logical pieces of code into your application.
- In simple terms, this class allows you to modularly plug any code into your application and automatically wires that component into the game loop!!
- Game.Compontents.Add(new Ship());

… DrawableGameComponent class

- This is essentially like the GameComponent but …
- In addition to the wiring of the initiaze() and update() methods in the game loop you also get the draw() method.

- In the case of Asteroids, I'm using this class because I want the SpriteManager to take care of drawing the sprites (all of them).

- In the initial game class when you create the instance of the SpriteManager (or any other GameComponent) don't forget to add it to the Components list :
  - Components.Add(spriteManager);

Missile Handling ...

- Design option ... but one possibility (amongst others)
- Is that each spaceship handles the drawing of missile fired.
- The missiles themselves then handle their lifetime ... Asking themselves ... Can I still hit something of have I run out of luck (screen space)
- Check the code for the SpaceShip + Missile sprites ...
- Really there are many alternatives. It’s up to you.

Randomness Factor

- In order to be an enjoyable game you need to put in some randomness factor ....
- In my case I've added random directions for the asteroids.
  - This is given by using a random initial rotation
- And random speeds to the asteroids as well.
  - This is given in the asteroid constructor:
  - asteroidVelocity = _maxVelocity * rand.Next(1,20);
- Originally there's also a random component in the size of the asteroids. I have not implemented that but it's easy to do following what we've already seen. Maybe it's more correct to talk about the ` probability of `

Rotations and Scaling

- When drawn sprites can be rotated and scaled ...
- In my implementation the spaceship is scaled by 0.16f
- Rotation is a floating number which is used to orient the spaceship ... However when we change the rotation we must make sure that the direction (after pressing thrust) changes as well!!
- Use can use sin(rotation) and cos(rotation) to get the x and y components of the direction.
- Check the code ....

Asteroids Game – Collisions

- When using sprites it is usually important to determine the occurrence of a collision between entities in you app.

- For eg. collisions in our game can occur between:
  - Spaceship and Spaceship
  - Spaceship and Asteroid
  - Spaceship and Missile
  - Missile and Asteroid
  - Missile and Spaceship

- If an asteroid hits another asteroid nothing happens ...

- If an bullet hits another bullet nothing happens ...

Handling Collision Detection

- In general, collision detection is the process of checking whether certain objects in your game have collided with other objects.
- Algorithms to do this can range from very simplistic to very complex.
- A combination of easy and hard collision detection algorithms is usually used, where the easy (less expensive) algorithm is used to check for intersection ... And if this reports True the finer (more compute intensive) and more accurate algorithm (per-pixel checking for eg) is then used.

Handling Collision Detection

- The basic algorithm involves looping through all the objects to check whether there are any collisions.
- This is obviously expensive. A better alternative would be to subdivide the space (2D in our case) and check only certain portions of the screen.
- In this game I'm simply using lists to store the objects but one can opt to use more efficient data structures (trees for example)

Distance between two points

- In order to determine whether two objects have collided I can checked the distance between the centres of the two objects which is equal to sqrt($(x1-x2)^2 + (y1-y2)^2$)
- Now I need to use a bounding area over my objects ... A circle of a rectangle.
- This will depend on the objects in the game ... If you want to check a ball for collision detection with something then use a circle.

Bounding Circles

- In Asteroids we would know the radius of the bounding circles ... We know these will not change during game play so we don't need to calculate the radius each time. It can be a constant for every object.

- Trivially two objects collide if r1 + r2 > D
- Probably more efficient using $r1^2+r2^2 > D^2$

- R1=radius of circle surrounding obj1
- R2=radius of circle surrounding obj2
- D=distance between centres of objects

Bounding Rectangles

- This is what I've used in my implementation.
- At every iteration of the game loop calculate the bounding rectangles of everything
  - Spaceships
  - Asteroids
  - Missiles
  - Bonuses
- Then check for intersections between these rectangles.
- XNA provides a BoundingRectangle class with an intersection test method.

Bounding Areas Precision

- In most of the cases there won't exist a geometric shape that will tightly fit your objects ....
- Therefore there's always the possibility of collision when in reality the objects do not collide.
- With Sprites one can improve on that by taking advantage of the image format of the texture which is usually a transparent image file (for ex png)
- When a collision is detected you can check whether the alpha component of the intersection pixels is equal to 0.
- This increases the overheads but gives you a more accurate collision detection algorithm

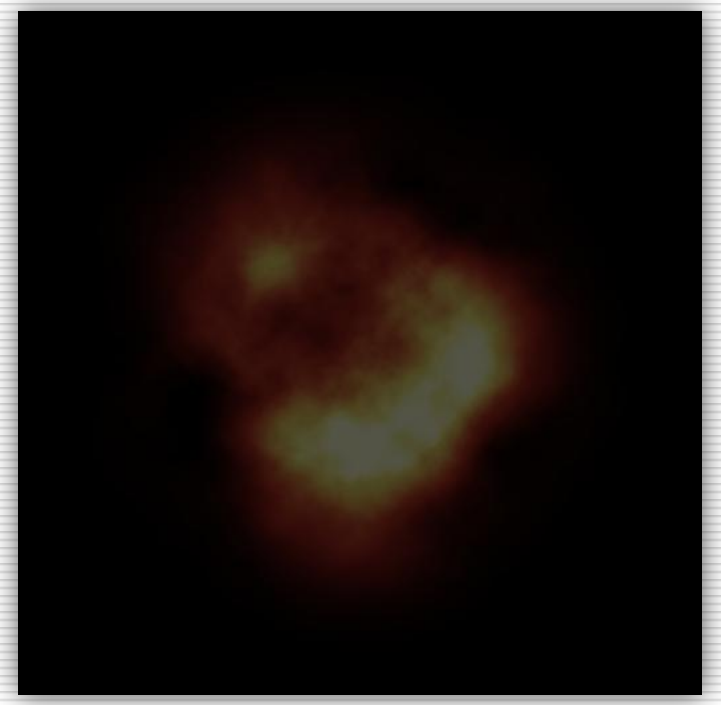
A Particle System (Simple) using Sprites

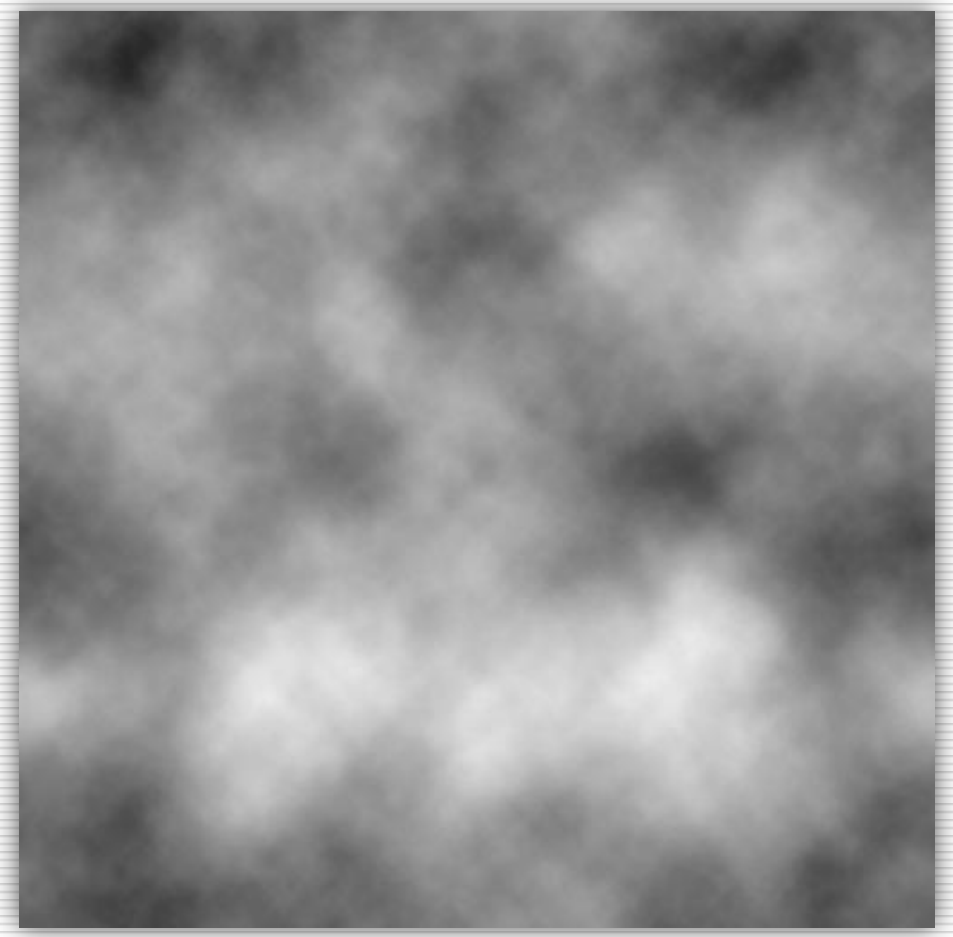
- Using sprites we can simulate a basic (but very effective in terms of visuals) particle system.
- Particle systems are generally used to simulate smoke, fire, flowing water (fountains), clouds, etc ...
- The basic idea to have a set of particles (sprites for this case) and animate them according to certain properties ....
- For example trying to simulate a particular wind direction ...
- Hence particles for us will be 2DTexture objects which we'll translate, rotate and scale across the screen.

Sprite properties … To represent particles …

- We need to define a core set of particle properties which we'll update with every frame.
- Moreover we need make use of the alpha channel of our 2DTexture objects in order to properly render (i.e. make realistic) the particle effects.
- Recall the alpha blending properties of texture namely:
  - Additive, add values of pixels.
  - AlphaBlend, use the underlying colour according to the alpha value of this pixel.
- These two properties are used to realistically render the particle effect when using sprite images.

Sample Textures for Smoke and Explosion





Rendering using the SpriteBatch ...

- public void Draw (Texture2D texture, Vector2 position, Nullable<Rectangle> sourceRectangle, Color color, float rotation, Vector2 origin, Vector2 scale, SpriteEffects effects, float layerDepth)

- *texture.* The sprite
- texture.*position.* The location, in screen coordinates, where the sprite will be drawn.
- *sourceRectangle.* A rectangle specifying, in texels, which section of the rectangle to draw. Use **null** to draw the entire texture.
- *color.* The color channel modulation to use. Use White for full color with no tinting.
- *rotation.* The angle, in radians, to rotate the sprite around the origin.
- *origin.* The origin of the sprite. Specify (0,0) for the upper-left corner.
- *scale.* Vector containing separate scalar multiples for the x- and y-axes of the sprite.
- *effects.* Rotations to apply before rendering.
- *layerDepth.* The sorting depth of the sprite, between 0 (front) and 1 (back). You must specify either FrontToBack or BackToFront for this parameter to affect sprite drawing.

Check Particle System Example

- Switch to Visual Studio Express and check the Project ParticleSampleWindows
- You can download this project from the XNA web site ...
- We shall look at this example and vary a number of parameters in order to understand how this effect works.
- Particle effects are not difficult to implement and can make a huge difference with the visuals of your application (game) ...
- Let's say you've the asteroids implementation we say last week ... You can try to combine the explosion (and smoke) effects into that game when the missiles hit the asteroids.
- This is all we'll cover on sprites ... As a final note about the SpriteBatch class you should know that this class encapsulates a vertex/pixel shader (like the BasicEffect class) which we'll be discussing (shaders) in a couple of weeks time.