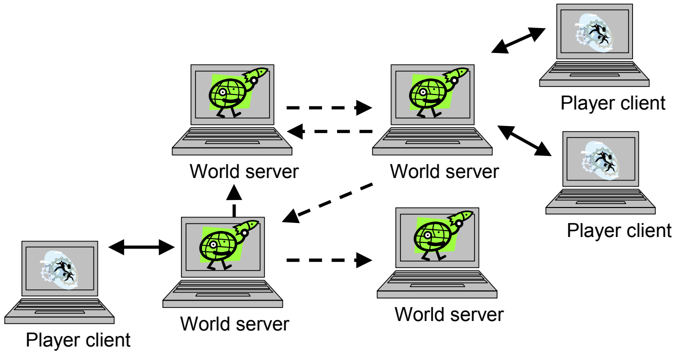
Gaggle topology



Worlds and Players

- A World server runs a game
- Players interact with the World to play in the game
- World can be connected so that Players can move from World to World
- The World controlled by a server is called that server’s Home World, to distinguish it from any other Worlds the server may know about
- The Player controlled by a client is called that client’s Home Player, to distinguish it from any other Players in the game

Worlds

- A World contains Players, Objects, and a Layout (the static walls, floors, and ceilings)
- A World implements a game by controlling how Players and other Objects interact with each other
- A World can connect to one or more other Worlds so that Players can move from World to World

Players

- A Player can be connected to at most one World at a time
- A Player interacts with a World by sending messages asking to do Actions, such as moving
- The World to which the Player is connected is the authority about the Player’s state, such as position and speed

Objects, Models, and Textures

- Objects are instances of Models and Textures
- Models are polygon meshes
- Texture are…textures
- Each Object has one Model and one Texture
- The same Model or Texture may be shared by multiple Objects
- Model and Texture do not form a pair. Two Object may share the same Model but have different Textures, or vice versa
- Models and Textures do not change over the life of the World. Objects do not change their Model or Texture during the life of the Object

Players and Objects

- Players are a type of object
- A Player has a Model and a Texture
- Players are indistinguishable from Objects to other Players in the World
- Players can ask the World about what Objects are in the World, what the Models and Textures are for those Objects
- Players can ask the World for the polygon description of each Model, and the image for each Texture in the World
- Players send their Model and Texture to the World when they join

Worlds and Objects

- Except for the Models and Textures provided by Players, the World is responsible for supplying Models and Textures
- On request, the World sends information about the Objects, Models, and Textures in the World
- A World can distinguish between Players and Objects, but on communicating Object info to Players, Players are treated like any other Object
- A World can create, move, and destroy Objects that are not Players
- A World can move any Player in the World. The World is in charge of the Player state, like it is in charge of all Objects

Models

- There are two formats of Models:
  - □ Static Models
  - □ MD2 Models
- Static Models are static vertex arrays, used for Objects which are not animated
- MD2 Models are animated, based on the MD2 format used in e.g. QuakeII, and can be loaded from disk

MD2 Models

- MD2 Models contain an array of vertex arrays
- One vertex array for every frame of animation
- The array of frames contains multiple animations
- What range of frames is used for what type of animation (e.g. walking, dying) is defined by the MD2 standard
- The World will inform the Player about what range of frames the animation should be looped over
- Two ways of rendering MD2 Models:
  - As a collection of individual triangles
  - Using the array of OpenGL “commands”

MD2 Models

First frame	Last frame	Anim type
0	39	Stand
40	45	Run
46	53	Attack
54	57	Pain A
58	61	Pain B
62	65	Pain C
66	71	Jump
72	83	Flip
84	94	Salute
95	111	Fall back
112	122	Wave
123	134	Point
135	153	Crouch stand
154	159	Crouch walk
160	168	Crouch attack
169	172	Crouch pain
173	177	Crouch death
178	183	Death fall back
184	189	Death fall forward
190	197	Death fall back slow

Rendering Objects

- Objects should be rendered based on their Model and Texture, the Object position and heading
- If the Model is an MD2Model, the appropriate animation frame should be used, possibly with interpolation
- There are exceptions to the position and heading requirements:
  - If the Object is fixed to the camera position, the Object is at the position of the camera (good for skyboxes)
  - If the Object is fixed to the camera rotation, the Object is rotated relative to the camera rotation (good for billboards)
  - If the Object is fixed to both the camera position and rotation, the Object is fixed to both the camera position and rotation (good for faking a HUD?)

Textures

- A Texture is a 2D array of RGB bytes
- Textures can be loaded from disk (uses the SDL_image library)
- Note that a texture may not be square or have a power of two size

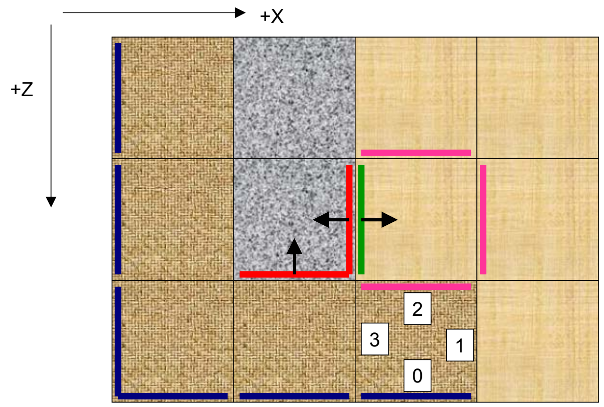
UIDs

- Each Object, Model, and Texture in a World has a unique ID (UID)
- UIDs are integers
- UIDs are only valid within the World that defined them
- Players ask a World about Objects, Models, and Textures using the UIDs
- A World is free to implement UIDs in whatever way they like, as long as the IDs are unique. Players can not make any assumption about how UIDs are assigned (i.e. don't use them as array indices)
- There is one special pre-define UID: UNASSIGNED_UID
  - Used when a UID has not been assigned yet, or the UID is not important
- When a World asks a Player about its Model or Texture, no UID is required (as a Player only has one Model and Texture). A UID is only needed for Objects, Models, and Textures which Players can ask about

World Layout

- The Layout is a 2D map describing where the walls, floors, and ceilings are in the World, and what their texture is
- The Layout is communicated to Players as a regular grid. It does not need to be implemented as such
- The Layout for a World can be any size and scale
- Players can ask about any rectangle of grid squares
- Worlds can reply with any rectangle of grid squares
- Each grid square has four walls, and may have a floor and a ceiling
- Walls are located on the square sides, the floor and ceiling extend over the entire square
- Walls can be partially or fully open
- Each wall, floor, and ceiling of a square can have its own texture
- Walls are single-sided, not shared between squares, and face inwards
- Walls in a square are numbered from 0 to 3, counter-clockwise starting at the +Z side

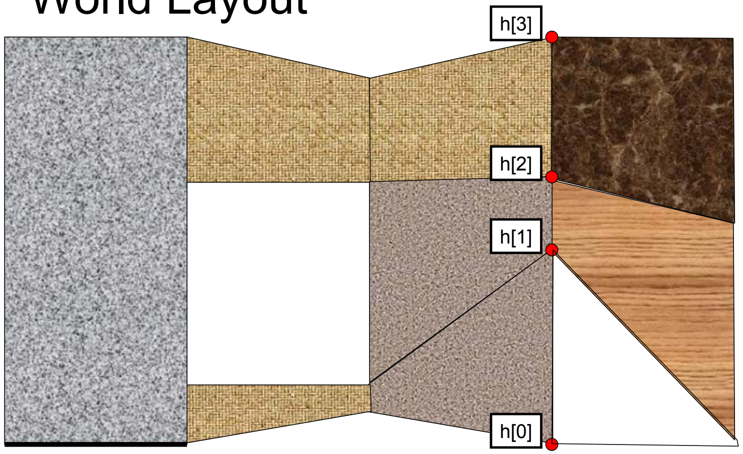
World Layout



World Layout

- Each wall is split into three sections stacked vertically
- Each wall section can be set to be open (see-through) or closed
- The Y coordinates of each wall section, floor, and ceiling is determined by heights specified at each grid point
- Each grid point has four heights:
  - Height of the floor
  - Height of the top of the first wall section
  - Height of the top of the second wall section
  - Height of the top of the third wall section and ceiling
- The heights at the grid points are shared among the grid squares

World Layout



World Layout

- The h[0] heights at the points of a grid square form the floor of that grid square
- The h[3] heights at the points of a grid square form the ceiling of that grid square
- A floor or ceiling may be omitted by setting its texture UID to UNASSIGNED_UID
- The textures on the walls, floors, and ceilings are repeated. The texture coordinates at each vertex is computed by a per square scale and offset applied to the vertex position in world coordinates (not grid coordinates). See the gaggle.h header file
- Each wall, floor, and ceiling can be brightened or darkened by a light value. How to use this light value is up to the renderer

Motion

- Objects have a 3D position in the World
- Objects have a heading in the World, with 0 pointing at +Z following the right hand screw rule (positive rotation is from +Z towards +X)
- Players request to be moved; Worlds tell the Players how they (and other Objects) are moving
- Each Object has a State, which defines where an Object is at a given time, and how it is expected to move over some period of time after that
- A State contains:
  - starttime, endtime : the time period over which the movement is made
  - pos : position at starttime
  - dpos : change in pos (i.e. velocity) at starttime
  - ddpos : change in dpos (i.e. acceleration) at starttime
  - heading : heading at starttime
  - dheading : change in heading at starttime
  - startframe : first animation frame to use for animated Models
  - endframe : last animation frame to use for animated Models
  - animfps : how many frames per second the animated Model animates at
  - animstarttime : the starting time of the animation

Motion

- The motion during the time period of a state is computed using the equation of motion with gravity:

  pos(t) = pos + dpos * dt + ½ * ddpos * dt^2

  dpos(t) = dpos + ddpos * dt

  heading(t) = heading + dheading * dt

  where dt is current time minus starttime
- If the current time is after the endtime, the endtime is used, effectively stopping the Object
- The state of an Object may be updated by the World before the current time reaches endtime
- All this is already implemented in gaggle_object.cpp

Messages

- Players and World communicate with each other through network messages
- Players can send messages to the World they are in. Worlds can send messages to Players playing in it, and to connected Worlds
- When a message arrives, it is unpacked into an appropriate structure or object
- Most messages will result in some method of HomeWorld or HomePlayer being called
- Other than the rendering, most of the work in this assignment is implementing those methods (see main.cpp for an example)
- See gaggle.h for the format of the messages
- Messages are one-way. They are not replied to directly
- Responses to message may not arrive in the same order as the original messages were sent (e.g. if you ask for a Model and then a Texture, you may first get a Texture back and then a Model)

Messages

- Messages sent by Worlds

JoinWorldMessage	to World to join
WelcomeWorldMessage	to World after receiving JoinWorldMessage
WelcomePlayerMessage	to Player when ready to start playing
ObjectStateMessage	to Players when an Object changes state
AskTextureMessage	to Player to ask for its Texture
TextureMessage	to Player after receiving AskTextureMessage
AskModelMessage	to Player to ask for its Model
ModelMessage	to Player after receiving AskModelMessage
ObjectsMessage	to Player after receiving AskObjectsMessage
TextMessage	to Players to send a text message
WorldIntroMessage	to Player after receiving JoinPlayerMessage
WorldLayoutMessage	to Player after receiving AskWorldLayoutMessage
ChangeWorldMessage	to Player to ask it to move to a different World
ScoreMessage	to Player to give some sort of scoring info

Messages

- Messages received by Worlds

JoinWorldMessage	from World asking to join handled by JoinPeerWorld() method
WelcomeWorldMessage	from World after sending JoinWorldMessage handled by JoinPeerWorld() method
PlayerActionMessage	from Player asking to change state handled by PlayerAction method()
AskTextureMessage	from Player asking for a texture handled by AskTexture() method
TextureMessage	from Player after sending AskTextureMessage handled by PlayerTexture() method
AskModelMessage	from Player asking for a model handled by AskModel() method
ModelMessage	from Player after sending AskModelMessage handled by PlayerModel() method
AskObjectsMessage	from Player asking for the Objects handled by AskObjects() method
TextMessage	from Player sending a text message handled by PlayerTextMessage()
AskWorldLayout	from Player asking for part of the World Layout handled by AskWorldLayout() method
PlayerReadyMessage	from Player indicating readiness to start playing

Messages

- Messages sent by Players

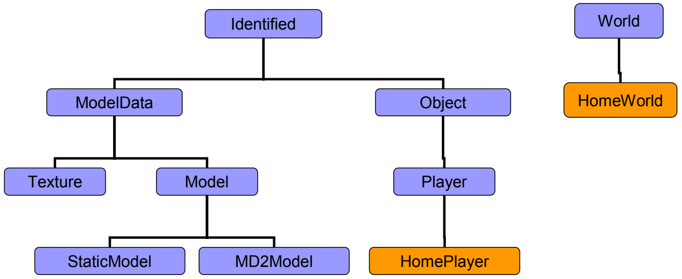
JoinPlayerMessage	to World asking to join
PlayerActionMessage	to World asking to change state
AskTextureMessage	to World asking for a Texture
TextureMessage	to World after receiving AskTextureMessage
AskModelMessage	to World asking for a Model
ModelMessage	to World after receiving AskModelMessage
AskObjectsMessage	to World asking for the Objects in World
TextMessage	to World giving it a text message
AskWorldLayoutMessage	to World asking for part of the World Layout
PlayerReadyMessage	to World indicating readiness to start playing

Messages

- Messages received by Players

□ WelcomePlayerMessage	from World when ready to start playing handled by JoinedWorld() method
□ ObjectStateMessage	from World when an Object changes state handled by ObjectState() method
□ AskTextureMessage	from World asking for Player’s Texture handled by AskTexture() method
□ TextureMessage	from World after sending AskTextureMessage handled by WorldTexture() method
□ AskModelMessage	from World asking for Player’s Model handled by AskModel() method
□ ModelMessage	from World after sending AskModelMessage handled by WorldModel() method
□ ObjectsMessage	from World after sending AskObjectsMessage handled by WorldObjects() method
□ TextMessage	from World giving us some text message handled by WorldTextMessage() method
□ WorldIntroMessage	from World after sending JoinPlayerMessage handled by WorldIntro() method
□ WorldLayoutMessage	from World after sending AskWorldLayoutMessage handled by WorldLayout() method
□ ChangeWorldMessage	from World asking Player to move to a different World handled by ChangeWorld() method
□ ScoreMessage	from World giving some sort of scoring info handled by WorldScoreMessage() method

Classes



Implementing a World server

- Sub-class HomeWorld to handle the response to various messages and events and keep the game state
- Probably sub-class Player to add whatever extra you need to handle Players in your HomeWorld
  - □ Create an instance and return it in JoinPlayer()
  - □ Gaggle passes that instance into methods dealing with Player messages
  - □ Destroy instance in PlayerHasLeft()
- Maybe sub-class World to add whatever extra you need to handle Worlds connected to your HomeWorld
  - □ Create instance in JoinPeerWorld()
  - □ Gaggle passes that instance into methods dealing with World messages
  - □ Destroy instance in LeftPeerWorld()

Implementing a Player client

- Sub-class HomePlayer to handle the response to various messages and events and keep track of the game state
- Sub-class StaticModel and MD2Model to include rendering methods
  - □ Make instances of sub-classes when model is received in WorldModel()
- Add user interaction subsystem, windowing subsystem, World Layout and Object rendering subsystem