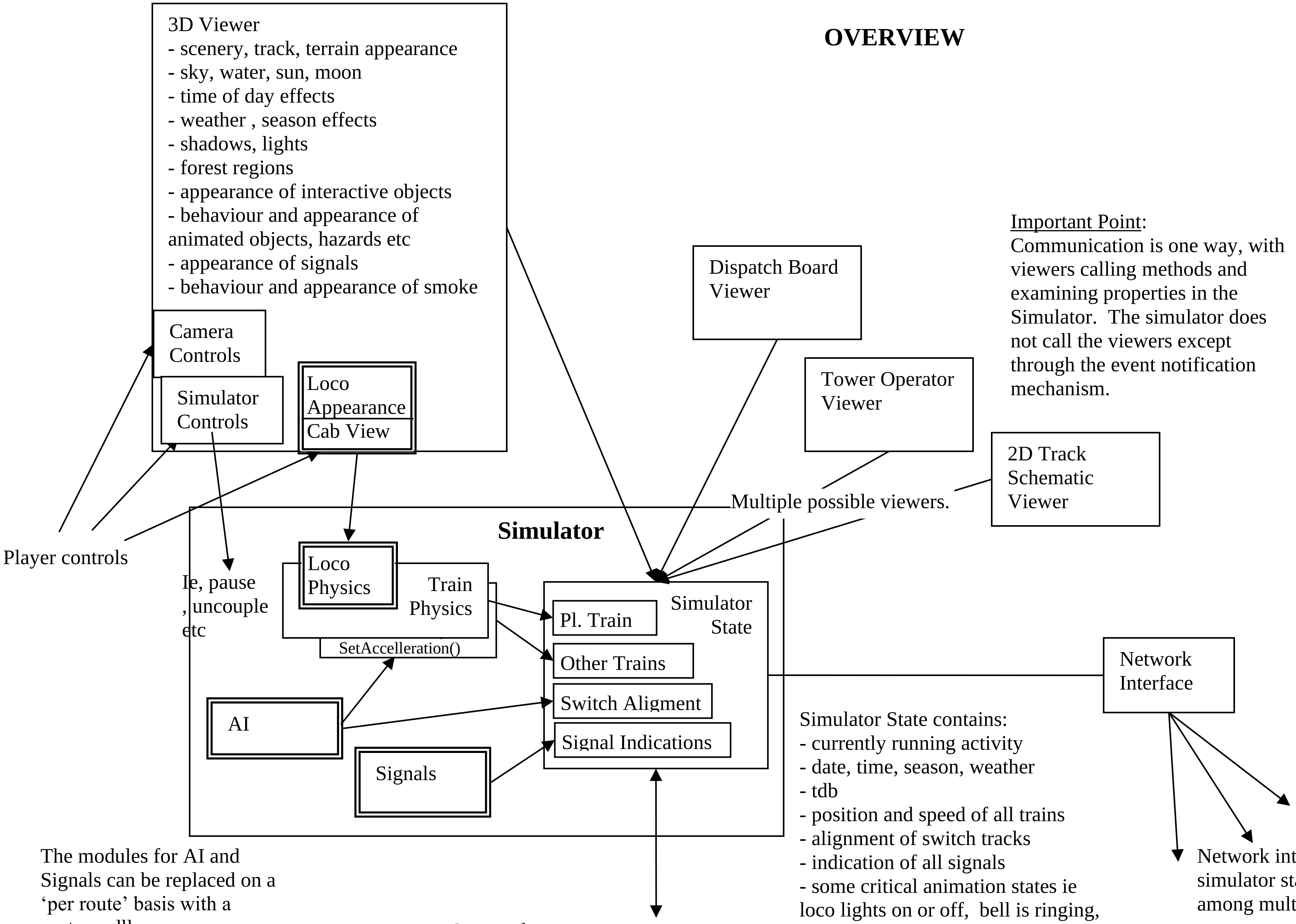
OVERVIEW



3D Viewer
- scenery, track, terrain appearance
- sky, water, sun, moon
- time of day effects
- weather , season effects
- shadows, lights
- forest regions
- appearance of interactive objects
- behaviour and appearance of animated objects, hazards etc
- appearance of signals
- behaviour and appearance of smoke

Important Point: Communication is one way, with viewers calling methods and examining properties in the Simulator. The simulator does not call the viewers except through the event notification mechanism.

Simulator State contains:
- currently running activity
- date, time, season, weather
- tdb
- position and speed of all trains
- alignment of switch tracks
- indication of all signals
- some critical animation states ie loco lights on or off, bell is ringing, pan is up or down

Network interface keeps simulator state in sync among multiple players

Remote computer does not run AI or Signals modules.

The modules for AI and Signals can be replaced on a ‘per route’ basis with a custom .dll

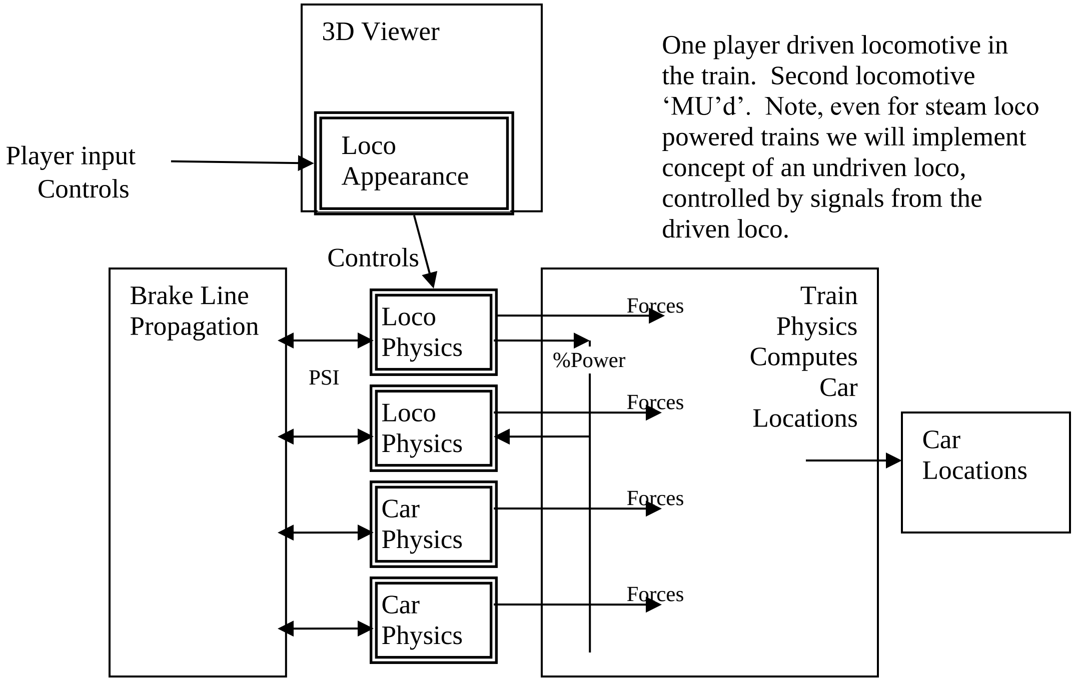
Loco Appearance and/or Loco Physics modules can be replaced on a ‘per wagon’ basis.

Save and restore games by saving and restoring simulator game state.

Since simulator state is kept in sync across the network, it should exclude info that can be regenerated locally – ie wheel rotation angle, smoke particles, scenery animation position, etc. These things will be maintained inside the 3D Viewer

PLAYER CONTROLLED TRAIN

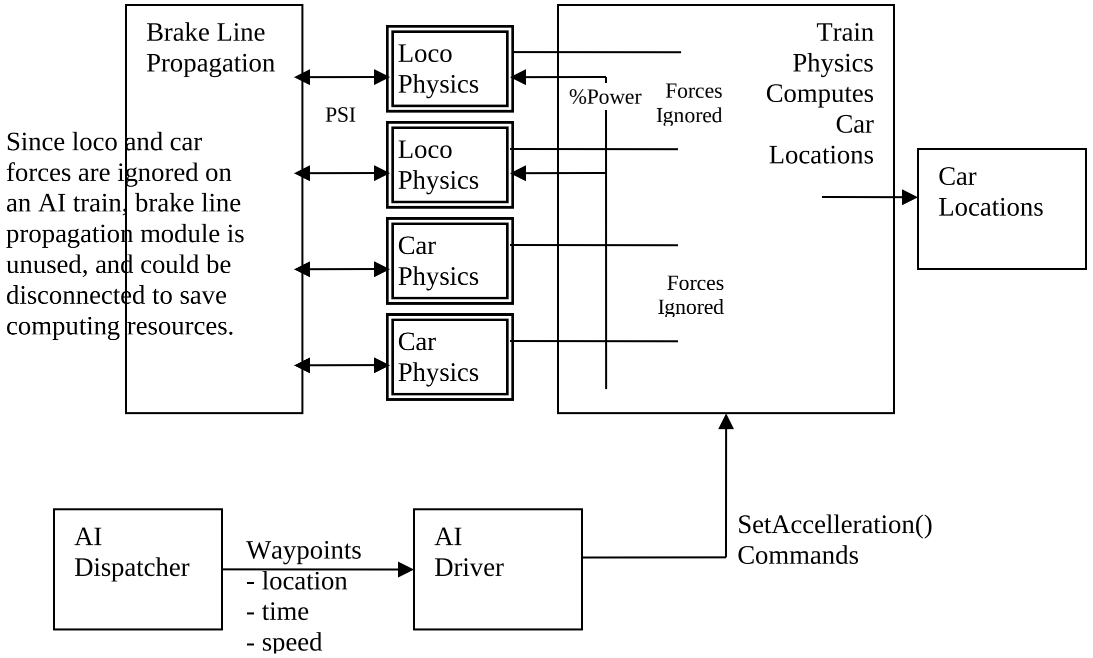
One player driven locomotive in the train. Second locomotive 'MU'd'. Note, even for steam loco powered trains we will implement concept of an undriven loco, controlled by signals from the driven loco.



A Loco Physics module must be able to respond to control commands from its matching 3D viewer module., OR, %power commands. The %power commands is the mechanism used to control undriven loco's in a multiple unit train, to control a loco on an AI driven train, and to control a loco that is driven remotely on another PC. Note that in the latter two cases, since the force calculations are ignored, the purpose of notifying the loco physics module of the loco's power setting is only to ensure its sounds, smoke generation etc are appropriate for the power levels it is running at.

AI CONTROLLED TRAIN

AI Controlled Train. The train physics module sends %power signals to physics module to control animation etc of the loco under power. However, forces are ignored. To ensure deterministic behaviour, train motion follows acceleration commands fromAI



Brake Line Propagation

PSI

Loco Physics

Loco Physics

Car Physics

Car Physics

%Power

Forces Ignored

Forces Ignored

Train Physics Computes Car Locations

Car Locations

Since loco and car forces are ignored on an AI train, brake line propagation module is unused, and could be disconnected to save computing resources.

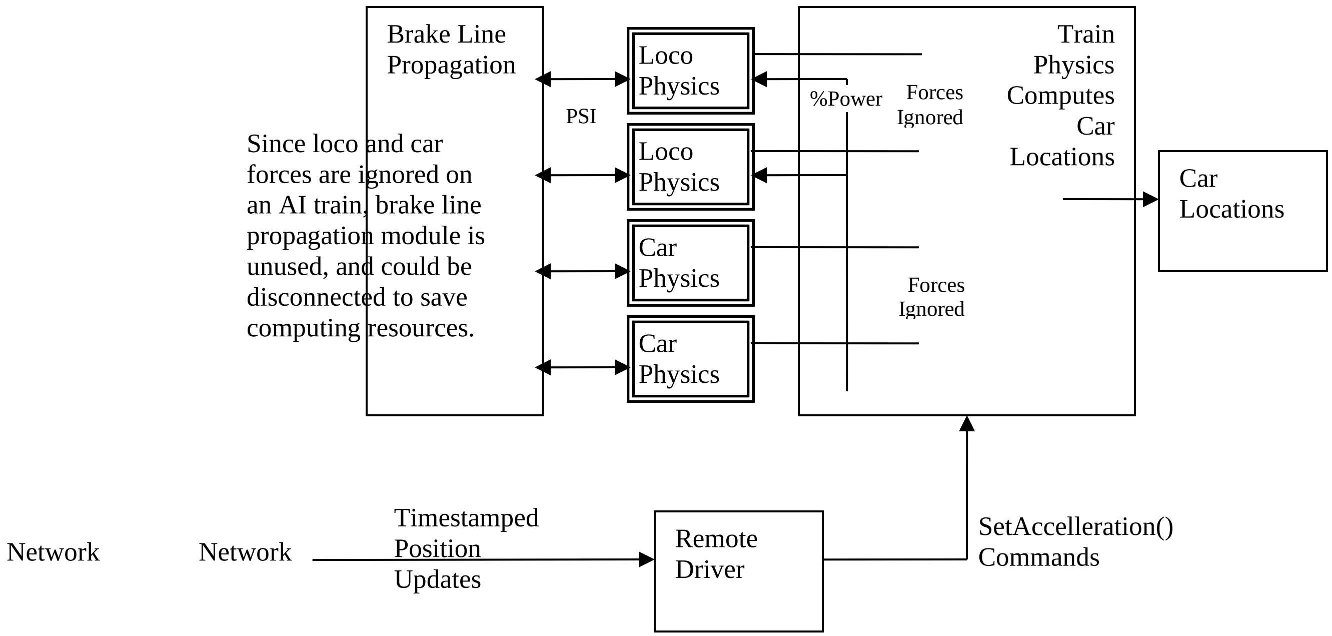
AI Dispatcher

Waypoints
- location
- time
- speed

AI Driver

SetAccelleration()
Commands

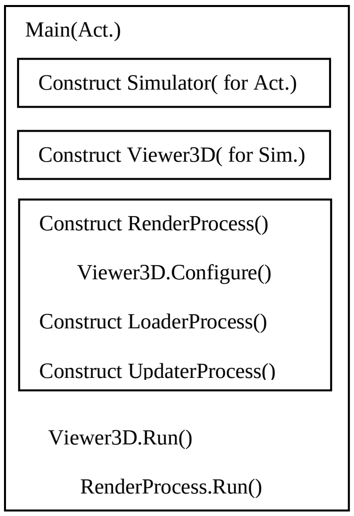
REMOTE CONTROLLED TRAIN



Since loco and car forces are ignored on an AI train, brake line propagation module is unused, and could be disconnected to save computing resources.

In a multiple player environment, each train is controlled by only one controlling computer. All other computers update the position of the train via the remote driver. The remote driver receives broadcasted timestamped position updates from the controlling computer. It compares these with where the train actually is on his computer, and issues setAccelleration commands as need .

STARTING PROGRAM



Main(Act.)

Construct Simulator(for Act.)

Construct Viewer3D(for Sim.)

Construct RenderProcess()

Viewer3D.Configure()

Construct LoaderProcess()

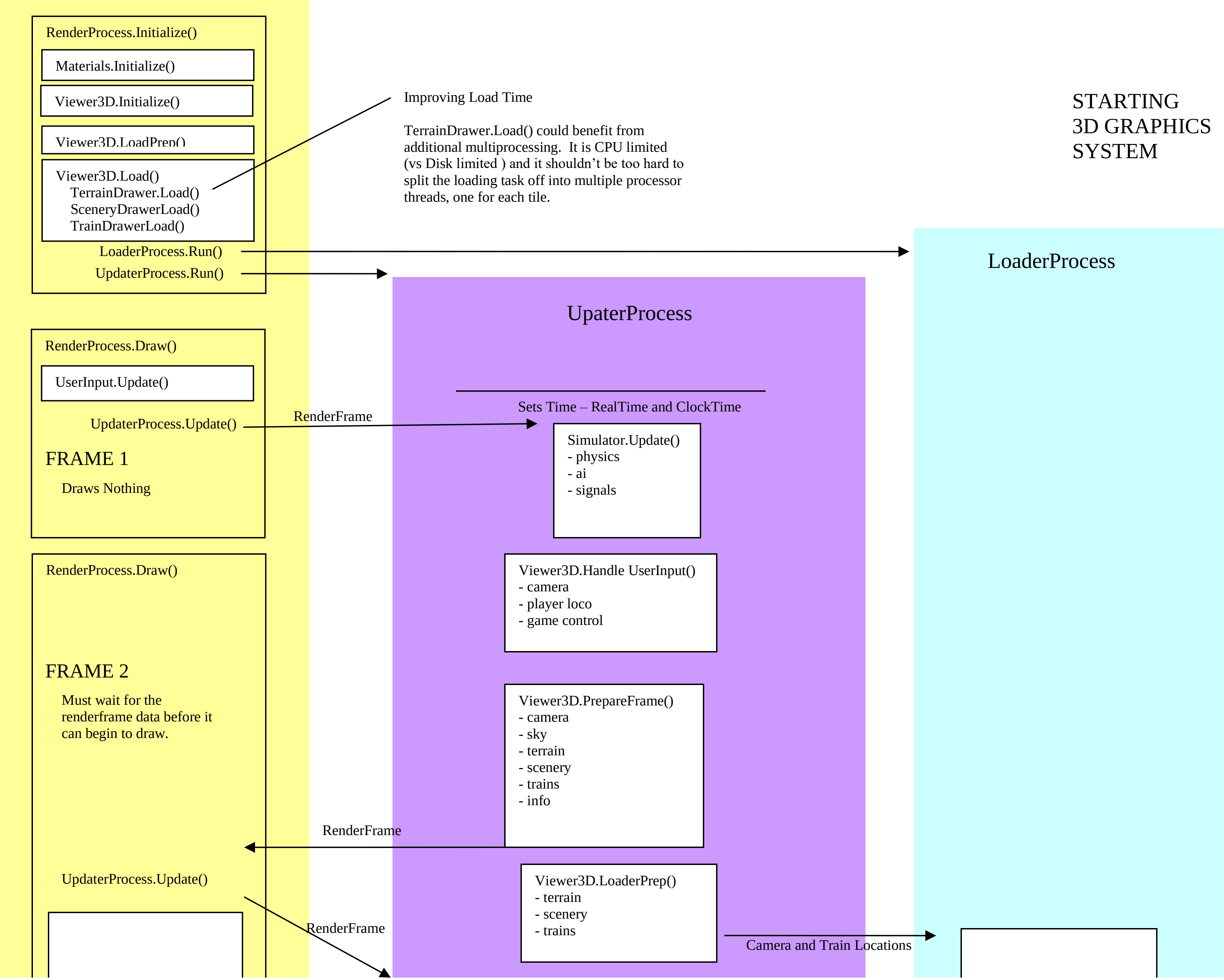
Construct UpdaterProcess()

Viewer3D.Run()

RenderProcess.Run()

RenderProcess
(XNA Game Class)

STARTING 3D GRAPHICS SYSTEM



RenderProcess.Initialize()

Materials.Initialize()

Viewer3D.Initialize()

Viewer3D.LoadPrep()

Viewer3D.Load()
TerrainDrawer.Load()
SceneryDrawerLoad()
TrainDrawerLoad()

LoaderProcess.Run()

UpdaterProcess.Run()

Improving Load Time

TerrainDrawer.Load() could benefit from additional multiprocessing. It is CPU limited (vs Disk limited) and it shouldn't be too hard to split the loading task off into multiple processor threads, one for each tile.

RenderProcess.Draw()

UserInput.Update()

UpdaterProcess.Update()

FRAME 1

Draws Nothing

RenderProcess.Draw()

FRAME 2

Must wait for the renderframe data before it can begin to draw.

UpdaterProcess.Update()

UpaterProcess

Sets Time – RealTime and ClockTime

RenderFrame

Simulator.Update()
- physics
- ai
- signals

Viewer3D.Handle UserInput()
- camera
- player loco
- game control

Viewer3D.PrepareFrame()
- camera
- sky
- terrain
- scenery
- trains
- info

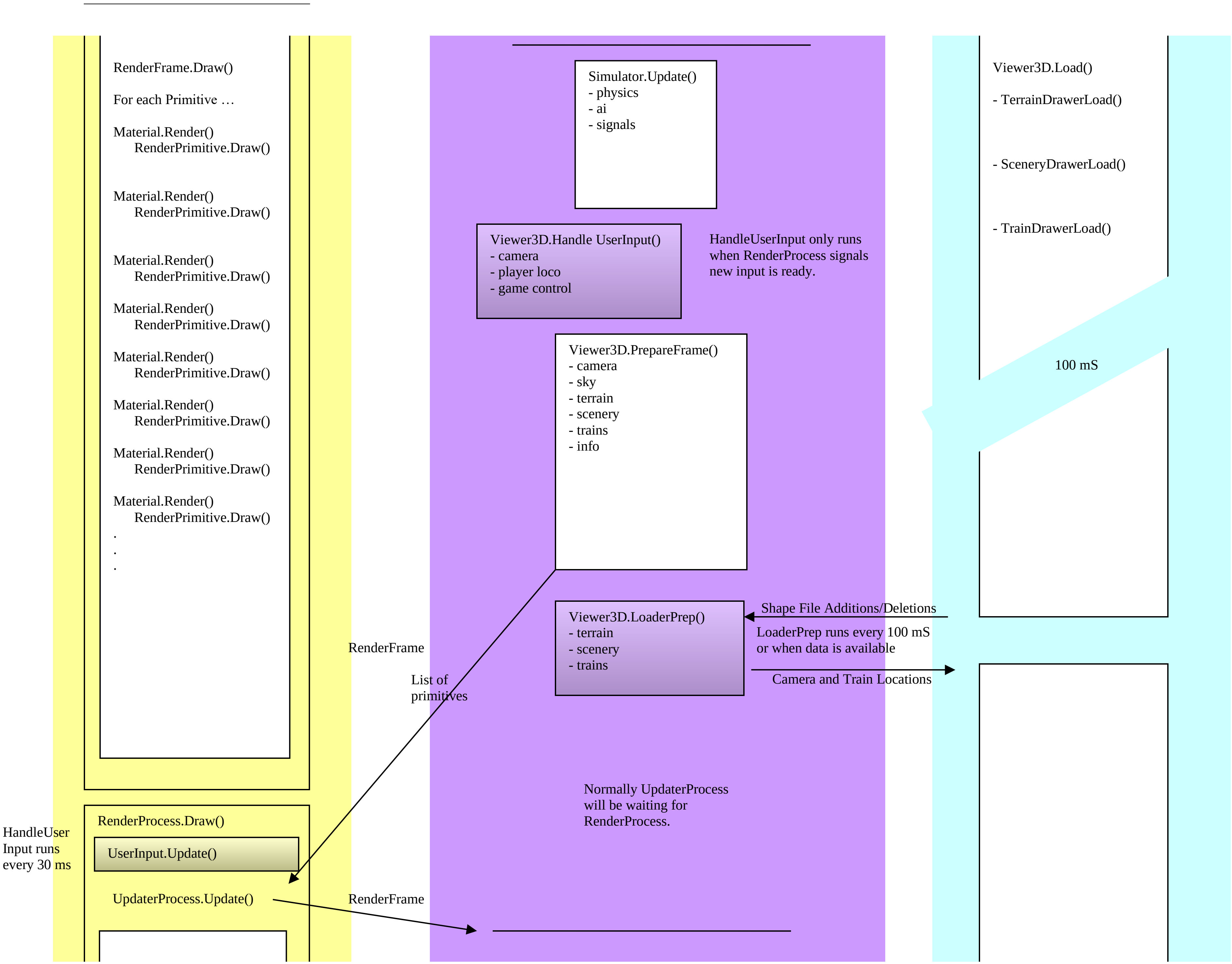
RenderFrame

Viewer3D.LoaderPrep()
- terrain
- scenery
- trains

RenderFrame

Camera and Train Locations

LoaderProcess



RenderFrame.Draw()

For each Primitive …

Material.Render()
RenderPrimitive.Draw()

Material.Render()
RenderPrimitive.Draw()

Material.Render()
RenderPrimitive.Draw()

Material.Render()
RenderPrimitive.Draw()

Material.Render()
RenderPrimitive.Draw()

Material.Render()
RenderPrimitive.Draw()

Material.Render()
RenderPrimitive.Draw()

Material.Render()
RenderPrimitive.Draw()
.
.
.

RenderProcess.Draw()

UserInput.Update()

UpdaterProcess.Update()

HandleUser Input runs every 30 ms

RenderFrame

List of primitives

RenderFrame

Simulator.Update()
- physics
- ai
- signals

Viewer3D.Handle UserInput()
- camera
- player loco
- game control

HandleUserInput only runs when RenderProcess signals new input is ready.

Viewer3D.PrepareFrame()
- camera
- sky
- terrain
- scenery
- trains
- info

Viewer3D.LoaderPrep()
- terrain
- scenery
- trains

Shape File Additions/Deletions

LoaderPrep runs every 100 mS or when data is available

Camera and Train Locations

Normally UpdaterProcess will be waiting for RenderProcess.

Viewer3D.Load()

- TerrainDrawerLoad()

- SceneryDrawerLoad()

- TrainDrawerLoad()

100 mS