

Technical Specification
Team 4: Image Space Inc.
CSE498, Fall 2004

George Gilbert
Paul Hoenecke
Adam Koby
Rob Sylvester



Table of Contents

Game Architecture

Architecture Overview	3
Camera System Architecture	3

New Camera Features

Vehicle Dependent Cameras	6
Camerman Error	8
Swingman Speed	8
Adding Roll to Swingman Camera	9
Add More Control to Swingman Camera	9
Water Ski Camera	9
Adding Transition Between Cameras	9

Function Listing

CameraMananger	10
BaseCameraManager	12
AttachedCameraManager	12
StaticCameraManager	14
TrackingCameraManager	15
isiBaseCam	16
isiLocalCam	17
isiStaticCam	18
isiTrackingCam	19
isiSplineData	20
isiSpline	20

Game Architecture

Architecture Overview

The game is divided into three stages: Initialization, Dynamic, and Post. The Initialization stage sets up all the game objects. This includes reading configuration files and making sure the game is ready to run. When the Initialization stage is complete, the game enters the dynamic stage. While in the dynamic state, the game loop is executed, as shown in figure 1. Every frame that is drawn is the result of one execution of this loop. Finally, when the game exits the dynamic stage, it enters the post stage. This stage is used to clean up game objects and free up used memory.

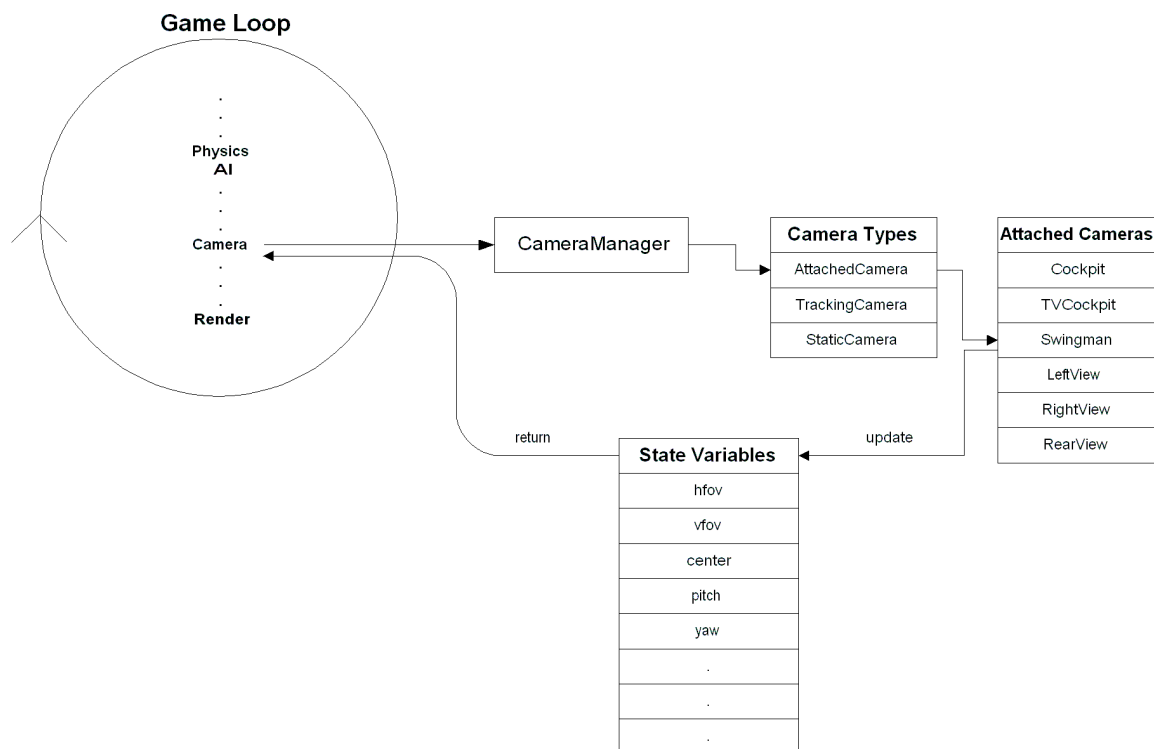


Figure 1

Camera System Architecture

Figure 1 demonstrates how the camera system fits into the games dynamic loop. Once the game has started running, the game loop executes once for every frame that is rendered. During this loop, the entire state of the game is calculated. These calculations are split up and performed by game subsystems, one of which is the camera system. The camera system uses the information calculated by the other systems to decide what should be shown on the screen. For example, if the camera is watching a computer

controlled vehicle that was moved by the AI system, the camera uses the vehicle's new position to change its orientation.

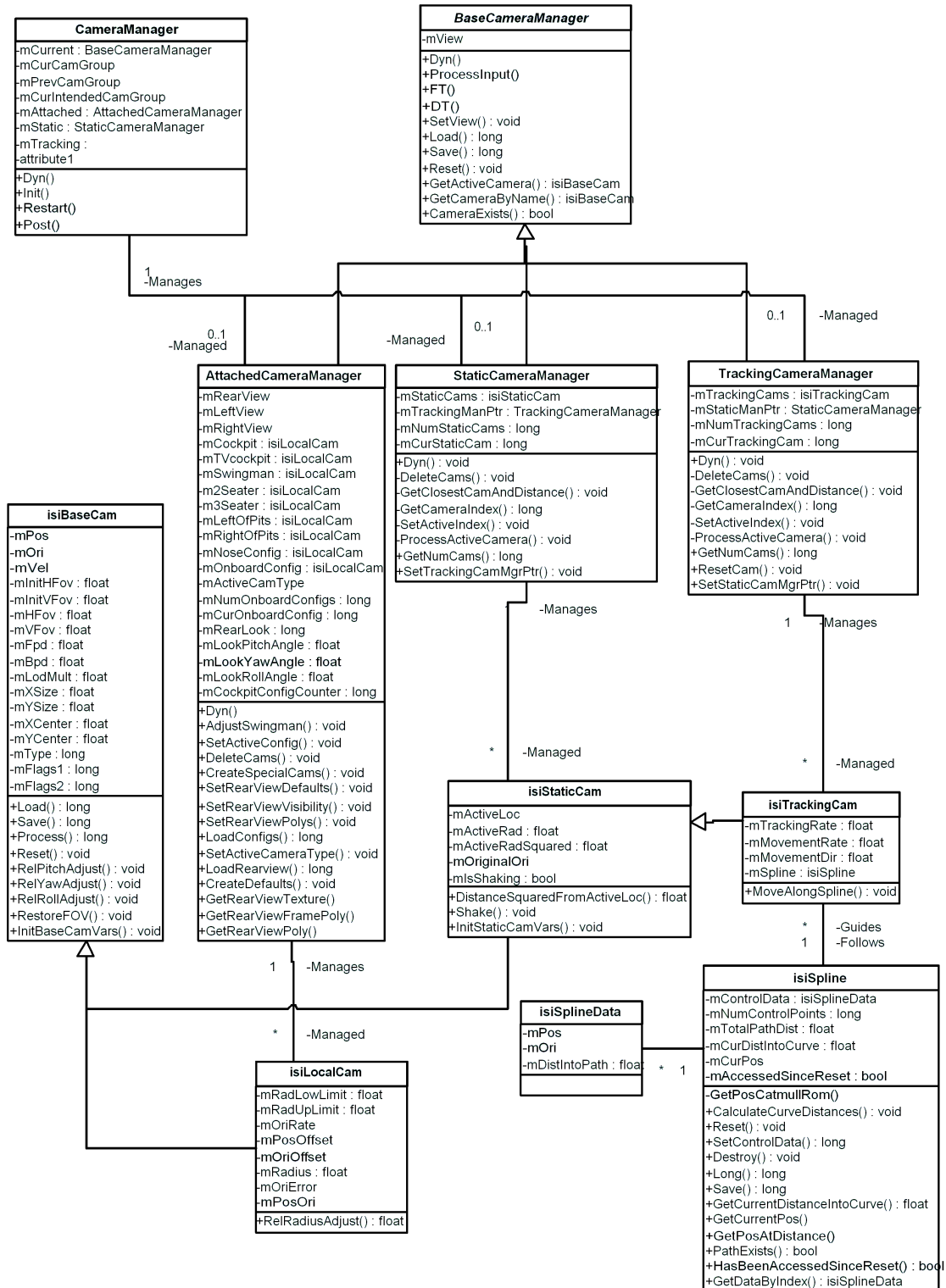


Figure 2

The camera system is comprised of several classes (see Figure 2) that work together to control what is drawn to the screen.

The classes `isiLocalCam`, `isiTrackingCam`, and `isiStaticCam` each contain the data associated with one specific camera. All cameras that are attached to a vehicle are of type `isiLocalCam`. `isiTrackingCam` and `isiStaticCam` are both trackside cameras. An `isiTrackingCam` has the ability to follow the movement of a target vehicle, whereas an `isiStaticCam` is not able to change orientation.

The classes `AttachedCameraManager`, `TrackingCameraManager`, and `StaticCameraManager` contain pointers to `isiLocalCams`, `isiTrackingCams`, and `isiStaticCams`, respectively. These classes control which of their cameras are currently active. Once the active camera is determined, it is updated to reflect the changes in the game.

The class `CameraManager` contains a pointer to an `AttachedCameraManager`, a `TrackingCameraManager`, and a `StaticCameraManager`. This class decides which group of cameras is currently active and then notifies the appropriate manager. The game contains a global instance of `CameraManager`, which is used to access the camera system from the game's dynamic loop.

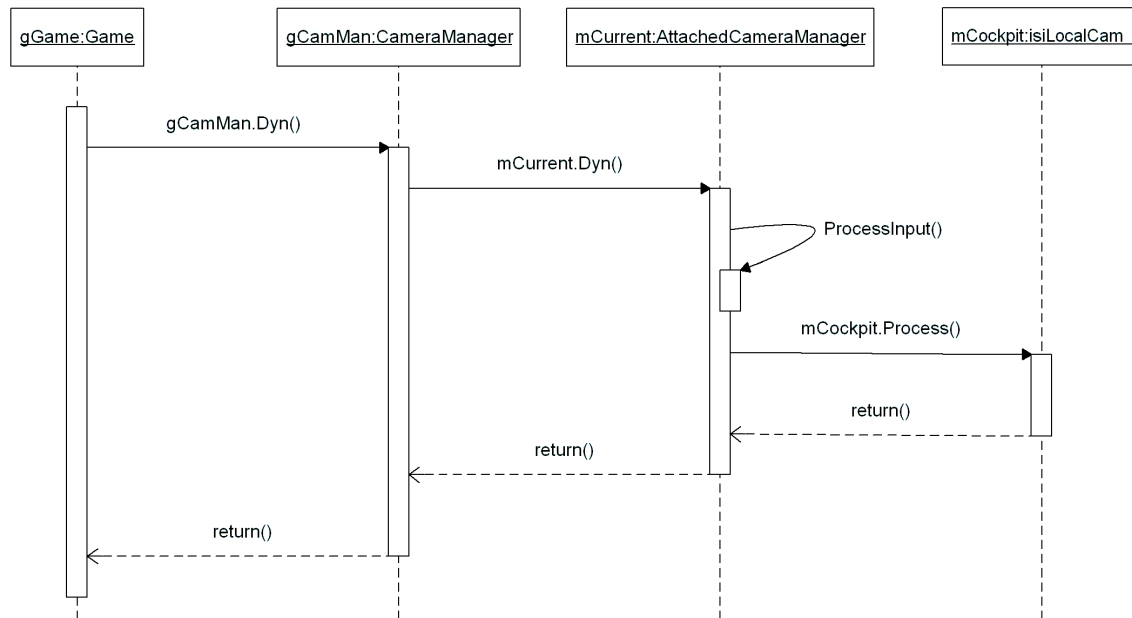


Figure 3

Figure 3 models the flow of control for the camera system during a typical execution of the game's dynamic loop. `gCamMan` is a global instance of `CameraManager`. `gGame` calls `gCamMan.Dyn()` to pass control to the camera system. In `gCamMan.Dyn()`, the Camera Manager determines which camera group is currently in use. In this case, the active camera group is the attached cameras. Therefore,

gCamMan's member variable mCurrent points to the AttachedCameraManager within gCamMan. Knowing this, the CameraManager calls mCurrent.Dyn(), passing control to the AttachedCameraManager. The AttachedCameraManager calls its member function ProcessInput(), which checks user input for changes to the currently active camera. In figure 3, mCockpit points to AttachedCameraManager's active camera, which is of type isiLocalCam. mCockpit.Process() is then called to apply the changes made to the camera. Once the changes are made, control is returned to gGame, which renders the scene and returns to the top of its dynamic loop.

New Camera Features

Vehicle Dependant Cameras

rFactor currently has only one set of attached cameras, which are used for all vehicles. These cameras are user configurable, but the camera properties remain the same even if the user switches vehicles. In previous Image Space Inc. games, this was not a problem because every vehicle had very similar dimensions (e.g. F1, NASCAR). However, in rFactor, this is not true. A camera that is set up for a smaller vehicle will be out of place if the user switches to a larger one.

A solution to this problem is to allow specific camera information for each vehicle. This would allow the designer of a vehicle to tailor the different cameras specifically to the new vehicle. Any vehicle without custom cameras defined would use default cameras; allowing backward compatibility with existing vehicles.

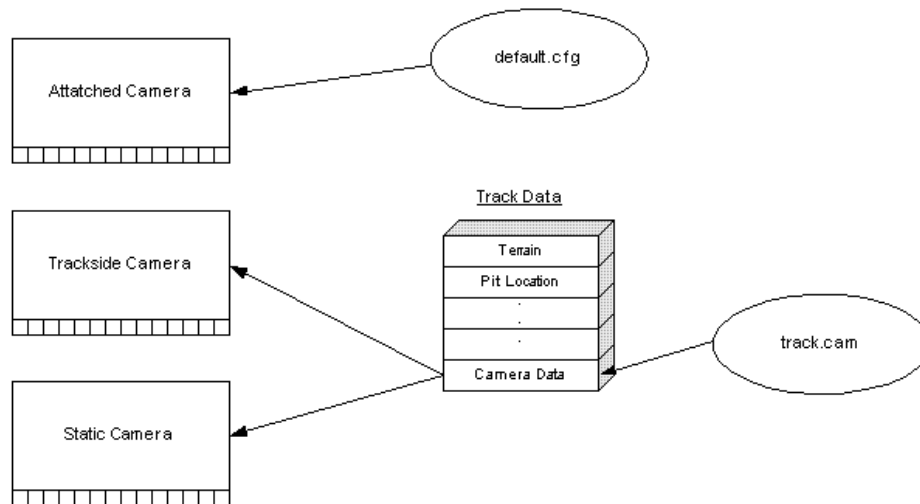


Figure 4

The cameras are currently created from user configurable settings defined in the file default.cfg (see Figure 4). These cameras are created only once and are never

changed. For any vehicle the user switches to, the same attached cameras will be used, whether they fit the new vehicle or not.

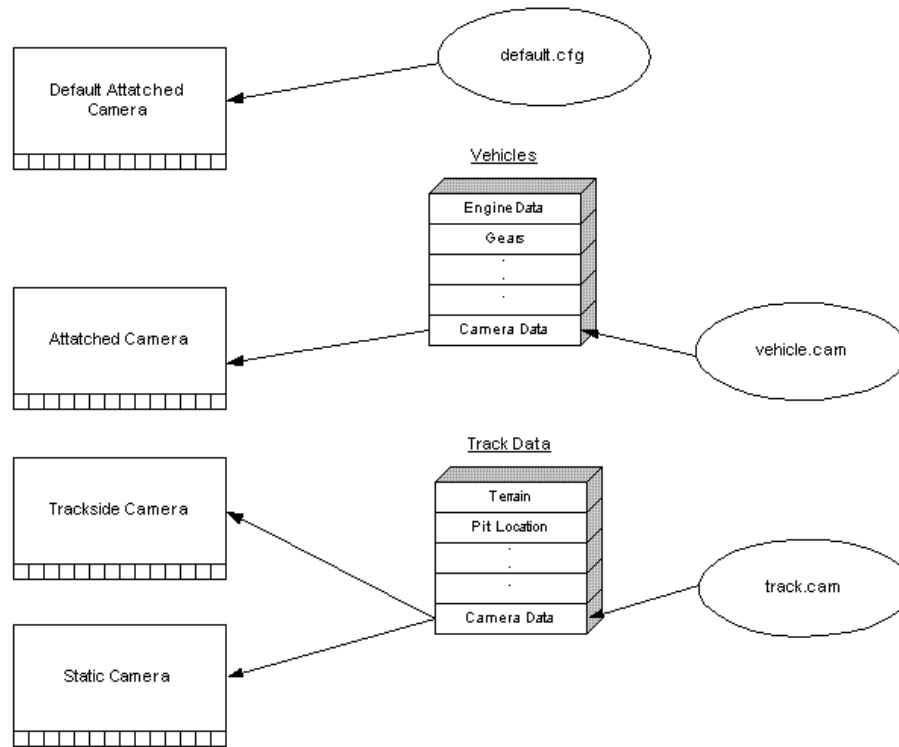


Figure 5

In the new system (Figure 5), the attached cameras will be created using data from the vehicle's configuration file. These vehicle specific cameras will be stored in the vehicle's data structure in the same format that AttachedCameraManager stores its cameras. When the user switches to a different vehicle, the stored camera data will be copied to the AttachedCameraManager, replacing the data from the last vehicle.

Below is a sample camera definition from the file default.cfg:

```
LocalCam=COCKPIT
{
  Fov=(75.000000, 62.500000)
  Clear=TRUE
  Color=(164, 218, 249)
  ClipPlanes=(0.225000, 500.000000)
  LODMultiplier=(1.000000)
  MipmapLODBias=(0.000000)
  Size=(1.000000, 1.000000)
  Center=(0.500000, 0.500000)
  Flags1=(0)
  Flags2=(0)
  RadiusLimits=(0.000000, 0.000000)
  OrientationRate=(999.000000, 999.000000, 999.000000)
  PositionOffset=(0.000000, 0.000, -0.0100000)
  OrientationOffset=(-0.030000, 0.000000, 0.000000)
  Radius=(0.000000)
}
```

Cameraman Error

Tracking cameras are cameras that follow a target vehicle from the side of the track. Currently the tracking cameras display a very exact behavior. In prior ISI games the cameras would keep the selected vehicle exactly in the center of the screen. Cameraman error has been added to some extent in rFactor, but needs improvement. Changes will be made to make a more humanistic appearance to how the cameras follow a vehicle.

- The camera should lose sight of the vehicle when the camera's turning speed is at its max. When this happens, the camera should try to reacquire the vehicle and possibly overcorrect while panning.
 - Currently, the tracking cameras have a maximum turning speed. If the turning speed is not fast enough to keep up with the vehicle, the camera will lose sight of it.
 - When this happens, the camera will decide whether to try to catch up fast, or continue turning at its normal speed. This decision will be made based on the size of the angle between the camera's orientation, and a vector from the camera to the vehicle. There will also be some randomness to the decision. The level of randomness will be configurable by the user.
 - If the camera decides to catch up fast, it should also decide whether it should overcorrect and go past the vehicle. This will also be based on some level of randomness configured by the user.
- A real life cameraman will try to keep the vehicle near the bottom of the screen. The trackside cameras should take this into account to add to the realistic look of the shot.
 - This can be accomplished by adjusting the pitch of the camera. The amount to adjust should be based on the angle between the camera's orientation and a vector from the camera to the vehicle. A level of randomness can also affect the magnitude of the change in pitch.
- The camera might zoom in fast on the vehicle if it is far away. Then as the vehicle approaches, the camera will zoom out.
 - The camera will first check the distance to the vehicle. Then, based on that distance and a user configurable level of randomness, the camera may decide to zoom in fast on the vehicle.
 - This check may not take place every frame.

Swingman Speed

The Swingman Camera is an attached camera that the user can zoom in and out, and rotate around the vehicle. Currently, the swingman camera can only move at one speed. The user should be able to set the speed at which the camera moves.

- The speed that the swingman camera moves is set in the code. The new version will allow the user to set a speed in the configuration file.

- There may be some acceleration to the movement of the camera. The user will be able to specify an acceleration factor in the configuration file.
 - The speed of the camera will increase based on the acceleration factor and the elapsed time. Once the speed reaches the maximum speed for the camera, it will stop accelerating.

Adding Roll to the Swingman Camera

Currently, the Swingman Camera does not roll when rotating around the vehicle. It would be “cool” if the camera would roll in the direction that the camera is rotating. The amount the camera rolls should be user configurable.

- To accomplish this effect, the camera will rotate around its orientation vector until it reaches a maximum roll angle.
- When the user stops rotating the camera, it should roll back into an upright position.

Add Control to Swingman Camera

The Swingman camera allows the user to rotate around the vehicle and zoom in and out. More control could be added to the swingman camera allowing the user to move the focus of the camera.

- The camera can only move a specified distance from the original position.
- The user can only shift the focus of the camera in the X and Z directions.

Water Ski Camera

The Water Ski camera would be a new camera view that simulates the effect of being towed by the vehicle. This camera will follow the vehicle but should swing out around corners.

- Should be implemented as an Attached Camera.
- The amount of swing should be user configurable.

Adding Transition between Cameras

When the user switches between attached cameras the view is changed instantaneously. A transition can be added to make camera changes smoother. When the camera is switched by the user, the camera would smoothly move to the position and orientation of the new camera.

- Only intended for attached cameras.
- Curves can be implemented using splines.
- May not be practical, due to clipping when moving through a car.
- The user should be able to turn this on and off.

Function Listing

CameraManager

void Init()

Initializes the CameraManager. Loads the static and tracking cameras from the track information. Called by the main game loop.

void Restart()

Reset the CameraManager's basic settings. Called by the main game loop.

void Dyn()

This function decides which is currently active and then tells that manager to process its changes. Called by the main game loop.

void Post()

Used to clean up the CameraManager when the game leaves its dynamic state. Called by the main game loop.

AttachedCameraManager *GetAttached()

Used to access the AttachedCameraManager.

isiCam::isiBaseCam* GetActiveCamera()

Used to access the currently active camera.

isiCam::isiBaseCam* GetCameraFromPF()

Returns a pointer to the Player's default camera.

const char * GetActiveCameraName()

Used to retrieve the name of the currently active camera.

isiCam::isiBaseCam* GetCameraByName(const char * camName)

Used to retrieve a camera by specifying the name of the camera wanted.

bool CameraExists(const isiCam::isiBaseCam * camToFind)

Used to determine whether or not the specified camera exists in the manager.

bool CameraExists(const char * camNameToFind)

Used to determine whether or not the specified camera exists in the manager.

VehicleVisibility CurrentVehicleVisibility()

Used to determine whether the player's vehicle should be visible.

cubeView* GetCurrentView()

Used to retrieve the current view.

cubeView* GetCamManView()

Used to retrieve the CameraManager's current view.

CameraGroup GetCurrentCameraGroup()

Used to access the currently active camera group.

void SetActiveCameraManager(BaseCameraManager *m)

Used to set which camera manager is currently active. The active manager is either an AttachedCameraManager, a StaticCameraManager, or a TrackingCameraManager.

long SetCurrentCameraGroup(CameraGroup g, bool allowAdvance = true)

Used to set the currently active camera group.

void ResetActiveCamera()

Used to reset the basic settings for the active camera.

void ResetActiveCameraManager()

Used to reset the active camera manager. The active manager is either an AttachedCameraManager, a StaticCameraManager, or a TrackingCameraManager.

void SetActiveCameraFromPF()

Used to set the active camera based on the default found in the Player File.

bool SetActiveCamera(const isiCam::isiBaseCam* newCam, bool reset = true)

Used to set the current camera using a pointer to the desired camera.

bool SetActiveCamera(const char * newCamName, bool reset = true)

Used to set the current camera by specifying the name of the desired camera.

bool CameraChangesAreLocked()

Used to determine if changes to the camera are currently allowed.

void SetExternalOverride (isiCam::isiBaseCam *extCam)

This function is used to set whether or not the camera manager is able to adjust camera settings.

void LockCameraChanges(bool lock)

Used to control whether or not the camera is locked to changes.

void SetSlotIndexOfTarget(long index)**void BounceTarget(bool forward)**

Used to change the index of the target camera.

void BounceTargetBySlot(bool forward)

Used to change the target index based on the slot.

void ResetTargetToPlayer()

Used to reset the current camera index back to the player's vehicle.

void EnterRandomCameraChangeMode(long minTime = 1L, long maxTime = 15L)

Used to begin a random switching between cameras.

void ExitRandomCameraChangeMode()

Used to stop randomly switching between cameras.

bool InRandomCameraChangeMode()

Used to determine whether or not the Random Camera Change Mode is active.

void EnterSpecialCameraMode(CameraGroup g)

Used to enter special camera modes, such as pit stop cameras.

void ExitSpecialCameraMode()

Used to exit the special camera modes.

void RestoreFromPitlaneCamRestrictions()**void SetFreeLook(bool newState)**

Used to turn Free Look on and off.

bool GetFreeLook() const

Used to determine whether or not Free Look is active.

void ZeroFreeLook()

Used to reset the Free Look angles to zero.

float FreeLookPitch() const

Used to access the pitch angle of Free Look.

float FreeLookYaw() const

Used to access the yaw angle of Free Look.

bool IsCurrentAttached()

Used to determine whether the current camera is attached or trackside.

bool InCockpitView()

Used to determine whether the current camera is the cockpit view.

bool InTvCockpitView()

Used to determine whether the current camera is the TV cockpit view.

bool InNoseCamView()

Used to determine whether the current camera is the nose cam view.

bool RearLooking()

Used to determine whether the current camera is rear looking.

BaseCameraManager**void SetView(cubeView *v)**

Sets the view data for the camera.

cubeView *PView()

Returns the view data for the camera.

Processing methods used by all cameras. These virtual functions must be overridden by the Camera Manager classes who inherit from BaseCameraManager.

virtual long Load(char *file)**virtual long Save(char *file, long createNew)****virtual void Dyn()****virtual void Reset()****virtual void ProcessInput()****virtual isiCam::isiBaseCam* GetActiveCamera()****virtual isiCam::isiBaseCam* GetCameraByName (const char * camName)****virtual bool CameraExists (const isiCam::isiBaseCam * camToFind)****virtual bool CameraExists (const char * camNameToFind)****float FT()**

Returns the time it takes to render one frame.

float DT()**AttachedCameraManager****void AdjustSwingman (long rad, long pitch, long yaw)**

Used to adjust the radius, pitch and yaw of a swingman camera.

void SetActiveConfig(AttachedCamType t)

Used to set the active camera configuration based on the type of camera.

void DeleteCams(ISITemplates::sList<long> *camList)

Used to remove cameras from the AttachedCameraManager.

void CreateSpecialCams()

Used to create some cameras that need special initialization.

void SetRearViewDefaults()

Used to set default values for the rear view.

void SetRearViewVisibility(bool on)

Used to turn the rear view on and off.

void SetRearViewPolys (bool includeSideViews=false)

long LoadConfigs(char *file);

Used to load the configurations of the cameras from a file.

void SetActiveCameraType(AttachedCamType t, long dir = 1, long forceSetting = 0)

Used to set the active camera type.

long Load(char *file)

Used to load the cameras from a file.

long Save(char *file, long createNew)

Used to save the cameras to a file.

long LoadRearview(char *file)

Used to load a rear view from a file.

void Dyn()

Used to process the current camera. Sets the position and orientation of the camera based on the movements of the car.

void Reset()

Used to reset the currently active camera.

void ProcessInput()

Used to process the input from the user.

isiCam::isiBaseCam* GetActiveCamera()

Used to access the currently active camera.

isiCam::isiBaseCam* GetCameraByType(AttachedCamType type)

Used to retrieve a camera by specifying the type of camera wanted.

isiCam::isiBaseCam* GetCameraByName(const char * camName)

Used to retrieve a camera by specifying the name of the camera wanted.

bool CameraExists(const isiCam::isiBaseCam * camToFind)

Used to determine whether or not the specified camera exists in the manager.

bool CameraExists(const char * camNameToFind)

Used to determine whether or not the specified camera exists in the manager.

bool SetActiveCamera(const isiCam::isiBaseCam* newCam, bool reset = true)

Used to set the current camera using a pointer to the desired camera.

bool SetActiveCamera(const char * newCamName, bool reset = true)

Used to set the current camera by specifying the name of the desired camera.

cubeView *PRearView()

Used to access the data for the rear view.

AttachedCamType GetActiveCamType()

Used to determine the type of the currently active camera.

void CreateDefaults();

Used to create default configurations for cameras that are not specified by the configuration file.

bool RearLooking()

Used to determine whether or not the rear view is on.

cubeTexture *GetRearViewTexture ()

Used to access the texture which stores what is seen in the rear view.

cubePolygon *GetRearViewFramePoly ()

cubePolygon *GetRearViewPoly ()

Used to access the polygon that the rear view texture is mapped on.

StaticCameraManager

void DeleteCams(ISITemplates::sList<long> *camList)

Used to remove cameras from the AttachedCameraManager.

void GetClosestCamAndDistance(Vect3& pos, long *index, float *dist)

Used to get the camera that is closest to the current camera. The closest camera may be a static or tracking camera.

long GetCameraIndex (const isiCam::isiBaseCam * cam)

Used to retrieve the index of the desired camera.

void SetActiveIndex(long index)

Used to set the active index.

void ProcessActiveCamera()

Used to process the active camera. This function updates the position and orientation of the camera, based on the changes in the game.

isiCam::isiBaseCam* GetCameraByName(const char * camName)

Used to retrieve a camera by specifying the name of the camera wanted.

bool CameraExists(const isiCam::isiBaseCam * camToFind)

Used to determine whether or not the specified camera exists in the manager.

bool CameraExists(const char * camNameToFind)

Used to determine whether or not the specified camera exists in the manager.

bool SetActiveCamera(const isiCam::isiBaseCam* newCam, bool reset = true)

Used to set the current camera using a pointer to the desired camera.

bool SetActiveCamera(const char * newCamName, bool reset = true)

Used to set the current camera by specifying the name of the desired camera.

long Load(char *file)

Used to load the cameras from a file.

long Save(char *file, long createNew)

Used to save the cameras to a file.

void Dyn()

Used to process the current camera. Sets the position and orientation of the camera based on the movements of the car.

void Reset()

Used to reset the currently active camera.

void ProcessInput()

Used to process the input from the user.

long GetNumCams()

Returns the number of static cameras managed by StaticCameraManager.

void SetTrackingCamMgrPtr(TrackingCameraManager *m)

In order to switch to the next closest camera on the track, which may be static or tracking, the StaticCameraManager must have a pointer to the TrackingCameraManager.

TrackingCameraManager

void DeleteCams(ISITemplates::sList<long> *camList)

Used to remove cameras from the AttachedCameraManager.

void GetClosestCamAndDistance(Vect3& pos, long *index, float *dist)

Used to get the camera that is closest to the current camera. The closest camera may be a static or tracking camera.

long GetCameraIndex (const isiCam::isiBaseCam * cam)

Used to retrieve the index of the desired camera.

void SetActiveIndex(long index)

Used to set the active index.

void ProcessActiveCamera()

Used to process the active camera. This function updates the position and orientation of the camera, based on the changes in the game.

isiCam::isiBaseCam* GetCameraByName(const char * camName)

Used to retrieve a camera by specifying the name of the camera wanted.

bool CameraExists(const isiCam::isiBaseCam * camToFind)

Used to determine whether or not the specified camera exists in the manager.

bool CameraExists(const char * camNameToFind)

Used to determine whether or not the specified camera exists in the manager.

bool SetActiveCamera(const isiCam::isiBaseCam* newCam, bool reset = true)

Used to set the current camera using a pointer to the desired camera.

bool SetActiveCamera(const char * newCamName, bool reset = true)

Used to set the current camera by specifying the name of the desired camera.

long Load(char *file)

Used to load the cameras from a file.

long Save(char *file, long createNew)

Used to save the cameras to a file.

void Dyn()

Used to process the current camera. Sets the position and orientation of the camera based on the movements of the car.

void Reset()

Used to reset the currently active camera.

void ProcessInput()

Used to process the input from the user.

isiCam::isiBaseCam* GetActiveCamera()

Returns the TrackingCameraManager's currently active camera.

long GetNumCams()

Returns the number of tracking cameras managed by TrackingCameraManager.

void ResetCam();

Used to reset the camera.

void SetStaticCamMgrPtr(StaticCameraManager *m)

In order to switch to the next closest camera on the track, which may be static or tracking, the TrackingCameraManager must have a pointer to the StaticCameraManager.

isiBaseCam

These functions must be implemented by classes inheriting from isiBaseCam.

virtual long Load(SmartFile& fp, char* name)

virtual long Save(SmartFile& fp, long indent)

virtual void RelPitchAdjust(bool up, long speed, float dt)

virtual void RelYawAdjust(bool up, long speed, float dt)

virtual void RelRollAdjust(bool up, long speed, float dt)

virtual long Process(Vect3* pos, Vect3* ori, Vect3* vel, float ft, float dt)

virtual void Reset()

void RestoreFOV()

Used to restore the field of view to the initial values.

void InitBaseCamVars()

This function initializes certain variables that must be set for the system to work.

char* GetName()

Used to retrieve the name of the camera.

Vect3* GetPos()

Used to retrieve the position of the camera.

Vect3* GetOri()

Used to retrieve the orientation vector of the camera.

Vect3* GetVel()

Used to retrieve the velocity vector of the camera

void GetInitFOV(float* hfov, float* vfov)

Used to retrieve the initial values for the field of view.

void GetFOV(float* hfov, float* vfov)

Used to retrieve the current values of the field of view.

void GetClipPlanes(float *f, float *b)

Used to retrieve the front and back clipping planes.

float GetLodMultiplier()

Used to retrieve the level of detail multiplier.

void GetSize(float *x, float *y)

Used to retrieve the size in the x and y directions.

void GetCenter(float *x, float *y)

Used to retrieve the center in the x and y directions.

float GetMipmapLodBias()

Used to retrieve the level of detail bias for mipmapping.

bool GetClearFlag()

Used to determine whether or not the camera should clear the background.

void GetColor(long *r, long *g, long *b)

Used to retrieve the background color in RGB.

long GetType()

Used to retrieve the type of camera.

long GetFlags1()

Used to retrieve special effect flags for the camera.

long GetFlags2()

Used to retrieve application specific flags for the camera.

void SetName(char* name)

Used to set the name of the camera.

void SetPos(Vect3* pos)

Used to set the position of the camera.

void SetOri(Vect3* ori)

Used to set the orientation vector of the camera.

void SetVel(Vect3* vel)

Used to set the velocity vector of the camera.

void SetInitFOV(float hfov, float vfov)

Used to set the initial field of view.

void SetFOV(float hfov, float vfov)

Used to set the current field of view.

void SetClipPlanes(float f, float b)

Used to set the front and back clipping planes.

void SetLodMultiplier(float m)

Used to set the level of detail multiplier.

void SetSize(float x, float y)

Used to set the size in the x and y directions.

void SetCenter(float x, float y)

Used to set the center in the x and y directions.

void SetMipmapLodBias(float bias)

Used to set the level of detail bias for mipmapping.

void SetClearFlag(bool c)

Used to set whether or not the background should be cleared.

void SetColor(long r, long g, long b)

Used to set the background color in RGB.

void SetType(long t)

Used to set the type of the camera.

void SetFlags1(long flags)

Used to set the special effect flags.

void SetFlags2(long special)

Used to set the application specific flags.

isiLocalCam**long Load(SmartFile& fp, char* name)**

Used to read the camera data from a file.

long Save(SmartFile& fp, long indent)

Used to save camera data to a file.

void RelPitchAdjust(bool up, long speed, float dt)

Used to adjust the pitch of the camera up or down.

void RelYawAdjust(bool up, long speed, float dt)

Used to adjust the yaw of the camera left or right.

void RelRollAdjust(bool up, long speed, float dt)

Used to adjust the roll of the camera in either direction.

long Process(Vect3* pos, Vect3* ori, Vect3* vel, float ft, float dt)

This function calculates changes to the camera for the current frame.

void Reset()

Used to reset the camera.

void RelRadiusAdjust(bool up, long speed, float dt)

Used to adjust the radius for a swingman camera.

void GetRadiusLimits(float* low, float* high)

Used to retrieve the upper and lower limits on the radius.

Vect3* GetOrientationRate()

Used to retrieve the orientation rate of the camera.

Vect3* GetPosOffset()

Used to retrieve the position offset for the camera.

Vect3* GetOrientationOffset()

Used to retrieve the orientation offset for the camera.

float GetRadius()

Used to retrieve the radius of a swingman camera.

Vect3* GetOrientationError()

Used to retrieve the orientation error.

Vect3* GetPositionalOrientation()

Used to retrieve the positional orientation of the camera.

void SetRadiusLimits(float low, float high)

Used to set the upper and lower limits for the radius.

void SetOrientationRate(Vect3* ori)

Used to set the rate that the orientation can change.

void SetPosOffset(Vect3* pos)

Used to set the position offset of the camera.

void SetOrientationOffset(Vect3* ori)

Used to set the orientation offset of the camera.

void SetRadius(float rad)

Used to directly set the radius of a swingman camera.

void SetPositionalOrientation(Vect3* ori)

Used to set the positional orientation of a camera.

isiStaticCam

long Load(SmartFile& fp, char* name)

Used to read the camera data from a file.

long Save(SmartFile& fp, long indent)

Used to save camera data to a file.

void RelPitchAdjust(bool up, long speed, float dt)

Used to adjust the pitch of the camera up or down.

void RelYawAdjust(bool up, long speed, float dt)

Used to adjust the yaw of the camera left or right.

void RelRollAdjust(bool up, long speed, float dt)

Used to adjust the roll of the camera in either direction.

long Process(Vect3* pos, Vect3* ori, Vect3* vel, float ft, float dt)

This function calculates changes to the camera for the current frame.

void Reset()

Used to reset the camera.

float DistanceSquaredFromActiveLoc(Vect3* pos)

Used to determine the distance squared from the current location to another position.

void Shake(float magnitude, float dt)

Used to make the camera shake.

void InitStaticCamVars()

Used to initialize variables that must be set.

Vect3* GetActivationLocation()

Used to retrieve the point used to activate the camera. Default location is the camera's position.

float GetActivationRadius()

Used to retrieve the radius inside which the camera is active.

float GetActivationRadiusSquared()

Used to retrieve the radius squared inside which the camera is active.

Vect3* GetOriginalOri()

Used to retrieve the original orientation of the camera.

bool IsShaking()

Used to determine whether or not the camera is currently shaking.

void SetActivationLocation(Vect3* l)

Used to set the activation location for the camera.

void SetActivationRadius(float r)

Used to set the activation radius for the camera.

void SetActivationRadiusSquared(float r)

Used to set the activation radius squared for the camera.

void SetOriginalOri(Vect3 *o)

Used to set the original orientation for the camera.

void SetIsShaking(bool s)

Used to turn shaking on or off.

isiTrackingCam

long Load(SmartFile& fp, char* name)

Used to read the camera data from a file.

long Save(SmartFile& fp, long indent)

Used to save camera data to a file.

void RelPitchAdjust(bool up, long speed, float dt)

Used to adjust the pitch of the camera up or down.

void RelYawAdjust(bool up, long speed, float dt)

Used to adjust the yaw of the camera left or right.

void RelRollAdjust(bool up, long speed, float dt)

Used to adjust the roll of the camera in either direction.

long Process(Vect3* pos, Vect3* ori, Vect3* vel, float ft, float dt)

This function calculates changes to the camera for the current frame.

void Reset()

Used to reset the camera.

void MoveAlongSpline(Vect3* pos, Vect3* vel, float dt)

Used to move along the spline, based on the current position, velocity and the time since the last move.

float GetTrackingRate()

Used to retrieve the tracking rate for the camera.

float GetMovementRate()

Used to retrieve the movement rate for the camera.

isiSpline& GetSpline()

Used to access the spline associated with the camera.

void SetTrackingRate(float r)

Used to set the tracking rate of the camera.

void SetMovementRate(float m)

Used to set the movement rate of the camera.

isiSplineData

Vect3* GetPos()

Used to retrieve the position of the point.

Vect3* GetOri()

Used to retrieve the orientation of the point.

float GetDist()

Used to retrieve the points distance along the path.

void SetPos(Vect3 *p)

Used to set the point's position.

void SetOri(Vect3 *o)

Used to set the point's orientation.

void SetDist(float d)

used to set the distance that the point is along the path.

isiSpline

Vect3* GetPosCatmullRom(float d)

Used to retrieve the point along the path, based on the distance along the path.

void CalculateCurveDistances()

Used to calculate the distance of the curve.

bool PathExists()

Used to determine whether or not a path exists, based on the control data.

void Reset()

Used to reset the current distance along the curve.

long SetControlData(Vect3 *list, long num)

Used to set the control points that define a curve.

void Destroy()

Used to destroy the current curve data.

long Load(SmartFile& fp)

Used to load the control point data from a file.

long Save(SmartFile& fp, long indent)

Used to save the control point data to a file.

float GetTotalPathDist()

Used to retrieve the total path distance.

long GetNumControlPoints()

Used to retrieve the number of control points that define the curve.

isiSplineData* GetDataByIndex(long index)

Used to retrieve a control point based on its index.

bool HasBeenAccessedSinceReset()

Used to determine whether or not the spline has been accessed since last reset.

long SetDataByIndex(long index, isiSplineData *d)

Used to set a control point based on the index.

float GetCurrentDistanceIntoCurve()

Used to retrieve the current distance into the curve.

Vect3* GetCurrentPos()

Used to retrieve the current position along the curve.

Vect3* GetPosAtDistance(float d)

Used to retrieve the point at distance d along the curve.