Joint Track Machine Learning

Generated by Doxygen 1.9.5

Chapter 1

Compile Instructions for Joint Track Machine Learning

1.1 External Libraries

1.1.1 Binary Downloads

- CUDA Toolkit 11.6 (`https://developer.nvidia.com/cuda-11-6-0-download-archive`)
  - I usually do the local installer so that I have everything that I need all at once. You will not need to compile this.
- Qt 5.15.2 (`https://www.qt.io/download`)
  1. "Go Open Source"
  2. Scroll down to "Looking for Qt Binaries" and hit "Download the Qt Online Installer"
  3. You'll want to downlaod the MSVC2019_64 version of Qt. There are some binaries for GCC and other versions of MSVC, but those can get beefy and we don't need them.
  4. Also make sure that you download the newest version of "designer". This gives you an gui framework to edit the JTML gui
- PyTorch "Libtorch" 1.12.1 (`https://pytorch.org/`)
  - Scroll down and select Stable->Windows->C++/Java->CUDA 11.6

1.1.2 To be Built

MAKE SURE YOU BUILD THE "RELEASE" NOT "DEBUG"

- VTK 7.1.1 (`https://vtk.org/download/`)
  - You'll download the zip of the source code.
  - You will build this with CMake
  - One thing that you'll need to do in the Visual Studio Project configuration is change "vtkRenderingLabel" to compile with MSVC2017.
  - Once you finish building it, make sure to "build" the "INSTALL" target.
- OpenCV 4.5.5 (`https://github.com/opencv/opencv/releases/tag/4.5.5`)
  - Download the zip of the source code and build using CMake. It shouldn't be too tough.

1.2 In-House Libraries

These need to be built in order using cmake.

1. CostFunctionTools (`https://github.com/ajensen1234/CostFunctionTools-CMake`)
    - You won't do any development on this, but you should clone it from github anyway.
    - Build this using CMake, you will need to link some of the previous ^^ external libraries
2. JTA_Cost_Functions (`https://github.com/ajensen1234/JTA_Cost_Functions-CMake`)
    - You will need to link to CostFunctionTools as well as the external libraries

1.3 External Library CMake Locations

- OpenCv— `path/to/build_dir`
- Pytorch— `/path_to_libtorch/share/cmake/Torch`
- Qt— `C:/Qt/5.15.2/msvc2019_64/lib/cmake/Qt5/`
- CUDA— CMake should find this automatically
- VTK– CMake should find this automatically.

1.4 Extras

You *might* need to download cuDNN separately (`https://developer.nvidia.com/cudnn`)

- Let me know if this doesn't work and I can send you the binaries and include files directly.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

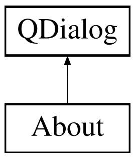
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 About Class Reference

Inheritance diagram for About:



Public Member Functions

- **About** (QWidget ∗parent=0, Qt::WindowFlags flags=0)
- void **setVersion** (int A, int B, int C)

The documentation for this class was generated from the following files:

- include/gui/about.h
- src/gui/about.cpp

5.2 Calibration Struct Reference

Public Member Functions

- **Calibration** (CameraCalibration monoplane_principal)
- Calibration (CameraCalibration biplane_A_principal, CameraCalibration biplane_B_principal, Vect_3 origin←_B, Matrix_3_3 axes_B)
- Matrix_3_3 **multiplication_mat_mat** (Matrix_3_3 X, Matrix_3_3 Y)
- Vect_3 **multiplication_mat_vec** (Matrix_3_3 X, Vect_3 u)
- Point6D **convert_Pose_A_to_Pose_B** (Point6D poseA)
- Point6D **convert_Pose_B_to_Pose_A** (Point6D poseA)

Public Attributes

- bool **biplane_calibration**
- CameraCalibration **camera_A_principal_**
- CameraCalibration **camera_B_principal_**
- Vect_3 **origin_B_**
- Matrix_3_3 **axes_B_**

5.2.1 Constructor & Destructor Documentation

5.2.1.1 Calibration()

```
Calibration::Calibration (
            CameraCalibration biplane_A_principal,
            CameraCalibration biplane_B_principal,
            Vect_3 origin_B,
            Matrix_3_3 axes_B )  [inline]
```

Parameters

biplane_A_principal	
biplane_B_principal	
origin_B	
axes_B	

The documentation for this struct was generated from the following file:

- include/core/calibration.h

5.3 camera_matrix Class Reference

Public Member Functions

- **camera_matrix** (Calibration calibration)

The documentation for this class was generated from the following files:

- include/core/camera_matrix.h
- src/core/camera_matrix.cpp

5.4 CameraCalibration Struct Reference

Public Member Functions

- **CameraCalibration** (float principal_distance, float principal_x, float principal_y, float pixel_pitch)

Public Attributes

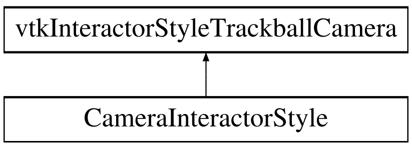
- float **principal_distance_**
- float **principal_x_**
- float **principal_y_**
- float **pixel_pitch_**

The documentation for this struct was generated from the following file:

- include/gpu/camera_calibration.h

5.5 CameraInteractorStyle Class Reference

Inheritance diagram for CameraInteractorStyle:



Public Member Functions

- **vtkTypeMacro** (CameraInteractorStyle, vtkInteractorStyleTrackballCamera)
- virtual void **OnChar** ()

Static Public Member Functions

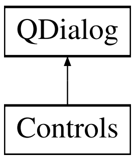
- static CameraInteractorStyle $*$ **New** ()

The documentation for this class was generated from the following file:

- include/gui/interactor.h

5.6 Controls Class Reference

Inheritance diagram for Controls:



Public Member Functions

- **Controls** (QWidget ∗parent=0, Qt::WindowFlags flags=0)

The documentation for this class was generated from the following files:

- include/gui/controls.h
- src/gui/controls.cpp

5.7 jta_cost_function::CostFunction Class Reference

Public Member Functions

- **__declspec** (dllexport) CostFunction()
- __declspec(dllexport) CostFunction(std **__declspec** (dllexport) ∼CostFunction()
- **__declspec** (dllexport) void addParameter(Parameter< double > new _parameter)
- **__declspec** (dllexport) void addParameter(Parameter< int > new _parameter)
- **__declspec** (dllexport) void addParameter(Parameter< bool > new _parameter)

The documentation for this class was generated from the following file:

- include/cost_functions/CostFunction.h

5.8 jta_cost_function::CostFunctionManager Class Reference

Public Member Functions

- **__declspec** (dllexport) CostFunctionManager(Stage stage)
- **__declspec** (dllexport) CostFunctionManager()
- **__declspec** (dllexport) ∼CostFunctionManager()
- __declspec(dllexport) void setActiveCostFunction(std __declspec(dllexport) bool updateCostFunction←ParameterValues(std __declspec(dllexport) bool updateCostFunctionParameterValues(std __declspec(dllexport) bool updateCostFunctionParameterValues(std __declspec(dllexport) bool InitializeActiveCostFunction(std __declspec(dllexport) bool DestructActiveCostFunction(std **__declspec** (dllexport) double callActiveCost←Function()
- __declspec(dllexport) std **__declspec** (dllexport) CostFunction ∗getActiveCostFunctionClass()
- __declspec(dllexport) CostFunction ∗getCostFunctionClass(std __declspec(dllexport) std **__declspec** (dll-export) void setCurrentFrameIndex(unsigned int current_frame_index)

The documentation for this class was generated from the following files:

- include/cost_functions/CostFunctionManager.h
- src/cost_functions/CostFunctionManager.cpp
- src/cost_functions/DD_NEW_POLE_CONSTRAINT.cpp
- src/cost_functions/DIRECT_DILATION.cpp
- src/cost_functions/DIRECT_DILATION_POLE_CONSTRAINT.cpp
- src/cost_functions/DIRECT_DILATION_SAME_Z.cpp
- src/cost_functions/DIRECT_DILATION_T1.cpp
- src/cost_functions/DIRECT_MAHFOUZ.cpp
- src/cost_functions/sym_trap_function.cpp

5.9 gpu_cost_function::CostFunctionToolboxGPU Class Reference

The documentation for this class was generated from the following file:

- include/gpu/gpu_toolbox.h

5.10 DirectDataStorage Class Reference

Public Member Functions

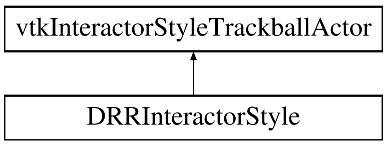
- **DirectDataStorage** (double initial_value)
- void **AddHyperBox** (HyperBox6D ∗new_box)
- void **DeleteHyperBoxes** (std::vector< int > col_ids)
- unsigned int **GetNumberColumns** ()
- HyperBox6D **GetMinimumHyperbox** (int col_id)
- double **GetMinimumHyperboxValue** (int col_id)
- double **GetSizeStoredInColumn** (int col_id)
- void **DeleteAllStoredHyperboxes** ()
- void **PrintSize** ()
- void **PrintContents** ()

The documentation for this class was generated from the following files:

- include/core/direct_data_storage.h
- src/core/direct_data_storage.cpp

5.11 DRRInteractorStyle Class Reference

Inheritance diagram for DRRInteractorStyle:



Public Member Functions

- **vtkTypeMacro** (DRRInteractorStyle, vtkInteractorStyleTrackballActor)
- void **initialize_DRRTool** (DRRTool ∗drrtool)
- bool **ActivePick** ()
- virtual void **OnChar** ()
- virtual void **OnKeyPress** ()
- virtual void **OnLeftButtonDown** ()
- virtual void **OnRightButtonDown** ()
- virtual void **OnMiddleButtonDown** ()
- virtual void **OnLeftButtonUp** ()
- virtual void **OnRightButtonUp** ()
- virtual void **OnMiddleButtonUp** ()
- virtual void **OnMouseMove** ()

Static Public Member Functions

- static DRRInteractorStyle ∗ **New** ()

Public Attributes

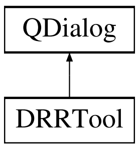
- DRRTool ∗ **drrtool_**

The documentation for this class was generated from the following file:

- include/core/drr_interactor.h

5.12 DRRTool Class Reference

Inheritance diagram for DRRTool:



Public Slots

- void **on_minLowerSpinBox_valueChanged** ()
- void **on_maxLowerSpinBox_valueChanged** ()
- void **on_minUpperSpinBox_valueChanged** ()
- void **on_maxUpperSpinBox_valueChanged** ()
- void **on_minSlider_valueChanged** ()
- void **on_maxSlider_valueChanged** ()

Public Member Functions

- **DRRTool** (Model model, CameraCalibration calibration, double model_z_plane, QWidget ∗parent=0, Qt::↩ WindowFlags flags=0)
- void **DrawDRR** ()

The documentation for this class was generated from the following files:

- include/gui/drr_tool.h
- src/gui/drr_tool.cpp

5.13 Frame Class Reference

Public Member Functions

- **Frame** (std::string file_location, int aperture, int low_threshold, int high_threshold, int dilation)
- cv::Mat **GetOriginalImage** ()
- cv::Mat **GetEdgeImage** ()
- cv::Mat **GetDilationImage** ()
- cv::Mat **GetInvertedImage** ()
- void **ResetFromOriginal** ()
- void **SetEdgeImage** (int aperture, int low_threshold, int high_threshold, bool use_reverse=false)
- void **SetDilatedImage** (int dilation)
- int **GetAperture** ()
- int **GetHighThreshold** ()
- int **GetLowThreshold** ()

Public Attributes

- std::string **file_location_**

The documentation for this class was generated from the following files:

- include/core/frame.h
- src/core/frame.cpp

5.14 HyperBox6D Struct Reference

Public Member Functions

- **HyperBox6D** (double value, Point6D center, Point6D sides)
- void **SetSides** (Point6D new_sides)
- Point6D **GetSides** ()
- bool **containsPoint** (Point6D point)
- Point6D **GetCenter** ()
- void **SetCenter** (Point6D new_center)
- void **TrisectSide** (Direction trisect_side)
- void **PrintCenter** ()

Public Attributes

- double **value_**
- double **size_**
- Point6D **sides_**
- Point6D **center_**

The documentation for this struct was generated from the following files:

- include/core/data_structures_6D.h
- src/core/data_structures_6D.cpp

5.15 HyperBoxGreaterThanSize Struct Reference

Public Member Functions

- bool **operator()** (const std::vector< HyperBox6D * > *old, double comparison)

The documentation for this struct was generated from the following file:

- src/core/direct_data_storage.cpp

5.16 HyperBoxLessThanValue Struct Reference

Public Member Functions

- bool **operator()** (const HyperBox6D *old, double comparison)

The documentation for this struct was generated from the following file:

- src/core/direct_data_storage.cpp

5.17 JTML_NFD Class Reference

Public Member Functions

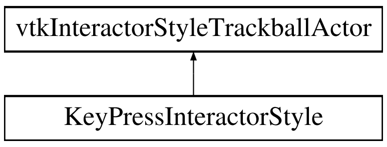
- bool **Initialize** (Calibration cal_file, std::vector< Model > model_list, std::vector< Frame > frames_list, QModelIndexList selected_models, unsigned int primary_model_index, QString error_message)
- void **Run** ()

The documentation for this class was generated from the following files:

- include/nfd/nfd.h
- src/nfd/nfd.cpp

5.18 KeyPressInteractorStyle Class Reference

Inheritance diagram for KeyPressInteractorStyle:



Public Member Functions

- **vtkTypeMacro** (KeyPressInteractorStyle, vtkInteractorStyleTrackballActor)
- void **initialize_MainScreen** (MainScreen ∗ms)
- bool **ActivePick** ()
- virtual void **OnChar** ()
- virtual void **OnKeyPress** ()
- virtual void **OnLeftButtonDown** ()
- virtual void **OnRightButtonDown** ()
- virtual void **OnMiddleButtonDown** ()
- virtual void **OnLeftButtonUp** ()
- virtual void **OnRightButtonUp** ()
- virtual void **OnMiddleButtonUp** ()
- virtual void **OnMouseMove** ()

Static Public Member Functions

- static KeyPressInteractorStyle ∗ **New** ()

Public Attributes

- MainScreen ∗ **ms_**

The documentation for this class was generated from the following file:

- include/gui/interactor.h

5.19 LocationStorage Class Reference

Public Member Functions

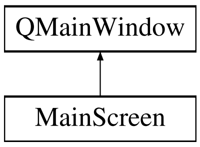
- void **LoadNewModel** (double principal_distance, double pixel_pitch)
- void **LoadNewFrame** ()
- Point6D **GetPose** (int frame_index, int model_index)
- void **SavePose** (int frame_index, int model_index, Point6D model_pose)
- int **GetFrameCount** ()
- int **GetModelCount** ()

The documentation for this class was generated from the following files:

- include/core/location_storage.h
- src/core/location_storage.cpp

5.20 MainScreen Class Reference

Inheritance diagram for MainScreen:



Public Slots

- void **optimizer_launch_slot** ()
- void **on_load_calibration_button_clicked** ()
- void **on_load_image_button_clicked** ()
- void **on_load_model_button_clicked** ()
- void **on_camera_A_radio_button_clicked** ()
- void **on_camera_B_radio_button_clicked** ()
- void **on_image_list_widget_itemSelectionChanged** ()
- void **on_model_list_widget_itemSelectionChanged** ()
- void **on_single_model_radio_button_clicked** ()
- void **on_multiple_model_radio_button_clicked** ()
- void **on_original_image_radio_button_clicked** ()
- void **on_inverted_image_radio_button_clicked** ()
- void **on_edges_image_radio_button_clicked** ()
- void **on_dilation_image_radio_button_clicked** ()
- void **on_original_model_radio_button_clicked** ()
- void **on_solid_model_radio_button_clicked** ()
- void **on_transparent_model_radio_button_clicked** ()
- void **on_wireframe_model_radio_button_clicked** ()
- void **on_aperture_spin_box_valueChanged** ()
- void **on_low_threshold_slider_valueChanged** ()
- void **on_high_threshold_slider_valueChanged** ()
- void **on_apply_all_edge_button_clicked** ()
- void **on_reset_edge_button_clicked** ()
- void **on_actionSave_Pose_triggered** ()
- void **on_actionSave_Kinematics_triggered** ()
- void **on_actionLoad_Pose_triggered** ()
- void **on_actionLoad_Kinematics_triggered** ()
- void **on_actionAbout_JointTrack_Auto_triggered** ()
- void **on_actionControls_triggered** ()
- void **on_actionStop_Optimizer_triggered** ()
- void **on_actionOptimizer_Settings_triggered** ()
- void **on_actionDRR_Settings_triggered** ()
- void **on_actionReset_View_triggered** ()
- void **on_actionReset_Normal_Up_triggered** ()
- void **on_actionModel_Interaction_Mode_triggered** ()
- void **on_actionCamera_Interaction_Mode_triggered** ()
- void **on_actionSegment_FemHR_triggered** ()
- void **on_actionSegment_TibHR_triggered** ()
- void **on_actionReset_Remove_All_Segmentation_triggered** ()
- void **on_actionEstimate_Femoral_Implant_s_triggered** ()

- void on_actionEstimate_Tibial_Implant_s_triggered ()

  Pose Estimate Progress and Label Visible/
- void on_actionNFD_Pose_Estimate_triggered ()

  Pose Estimate Progress and Label Visible/
- void **on_actionCopy_Next_Pose_triggered** ()
- void **on_actionCopy_Previous_Pose_triggered** ()
- void **on_actionLaunch_Tool_triggered** ()
- void **on_actionAmbiguous_Pose_Processing_triggered** ()
- void **on_optimize_button_clicked** ()
- void **on_optimize_all_button_clicked** ()
- void **on_optimize_each_button_clicked** ()
- void **on_optimize_from_button_clicked** ()
- void **on_actionOptimize_Backward_triggered** ()
- void **onUpdateOptimum** (double, double, double, double, double, double, unsigned int)
- void **onOptimizedFrame** (double, double, double, double, double, double, bool, unsigned int, bool, QString)
- void **onOptimizerError** (QString error_message)
- void **onUpdateDisplay** (double, int, double, unsigned int)
- void **onUpdateDilationBackground** ()
- void **updateOrientationSymTrap_MS** (double, double, double, double, double, double)
- void **onSaveSettings** (OptimizerSettings, jta_cost_function::CostFunctionManager, jta_cost_function::CostFunctionManager, jta_cost_function::CostFunctionManager)

Signals

- void **UpdateDisplayText** (bool)
- void **StopOptimizer** ()
- void **UpdateTimeRemaining** (int)

Public Member Functions

- MainScreen (QWidget ∗parent=0)
- void **VTKEscapeSignal** ()
- void **VTKMakePrincipalSignal** (vtkActor ∗new_principal_actor)

Public Attributes

- bool **currently_optimizing_**

Protected Member Functions

- void **resizeEvent** (QResizeEvent ∗event)
- void **keyPressEvent** (QKeyEvent ∗event)

5.20.1 Constructor & Destructor Documentation

5.20.1.1 MainScreen()

```
MainScreen::MainScreen (
            QWidget * parent = 0 )
```

Set up VTK/

5.20.2 Member Function Documentation

5.20.2.1 on_actionEstimate_Tibial_Implant_s_triggered

```
void MainScreen::on_actionEstimate_Tibial_Implant_s_triggered ( )  [slot]
```

Pose Estimate Progress and Label Visible/

Segment/ STL Information/ GPU Models for the current Model/ Load JIT Model/ Load JIT Z Model/ ∗Send Each Segmented Image to GPU Tensor, Predict Orientation, Then Z (From Area), then X,Y. *Flip Segment/ Render/ Copy To Mat/ OpenCV Image Container/Write Function/ Get Scale/ Get X and Y/ Convert from (0,0) Centered/ Update Model Pose/ Delete GPU Model/ Free Array/ Free Values/ Update Model/ Pose Estimate Progress and Label Not Visible/*

5.20.2.2 on_actionNFD_Pose_Estimate_triggered

```
void MainScreen::on_actionNFD_Pose_Estimate_triggered ( )  [slot]
```

Pose Estimate Progress and Label Visible/

Segment/ STL Information/ GPU Models for the current Model/ Load JIT Model/ Load JIT Z Model/ ∗Send Each Segmented Image to GPU Tensor, Predict Orientation, Then Z (From Area), then X,Y. *Flip Segment/ Render/ Copy To Mat/ OpenCV Image Container/Write Function/ Get Scale/ Get X and Y/ Convert from (0,0) Centered/ Update Model Pose/ Delete GPU Model/ Free Array/ Free Values/ Update Model/ Pose Estimate Progress and Label Not Visible/ Pose Estimate Progress and Label Visible/ Segment/ STL Information/ GPU Models for the current Model/ Load JIT Model/ Load JIT Z Model/* ∗Send Each Segmented Image to GPU Tensor, Predict Orientation, Then Z (From Area), then X,Y. *Flip Segment/ Render/ Copy To Mat/ OpenCV Image Container/Write Function/ Get Scale/ Get X and Y/ Convert from (0,0) Centered/ Update Model Pose/ Delete GPU Model/ Free Array/ Free Values/ Update Model/ Pose Estimate Progress and Label Not Visible/*

The documentation for this class was generated from the following files:

- include/gui/mainscreen.h
- src/gui/mainscreen.cpp

5.21 Matrix_3_3 Struct Reference

Public Member Functions

- **Matrix_3_3** (float A_11, float A_12, float A_13, float A_21, float A_22, float A_23, float A_31, float A_32, float A_33)
- Matrix_3_3 **tranpose** ()

Public Attributes

- float **A_11_**
- float **A_12_**
- float **A_13_**
- float **A_21_**
- float **A_22_**
- float **A_23_**
- float **A_31_**
- float **A_32_**
- float **A_33_**

The documentation for this struct was generated from the following file:

- include/core/calibration.h

5.22 Model Class Reference

Public Member Functions

- **Model** (std::string file_location, std::string model_name, std::string model_type)

Public Attributes

- std::string **file_location_**
- vtkSmartPointer< vtkSTLReader > **cad_reader_**
- std::vector< float > **triangle_vertices_**
- std::vector< float > **triangle_normals_**
- std::string **model_name_**
- std::string **model_type_**
- bool **initialized_correctly_**

The documentation for this class was generated from the following files:

- include/core/model.h
- src/core/model.cpp

5.23 nfd_instance Class Reference

Public Member Functions

- **nfd_instance** (GPUModel &gpu_mod, float xt, float yt, float zt, float xr, float yr, float zr)
- void **print_contour_points** ()
- void **print_raw_points** ()

The documentation for this class was generated from the following files:

- include/nfd/nfd_instance.h
- src/nfd/nfd_instance.cpp

5.24 nfd_library Class Reference

Public Member Functions

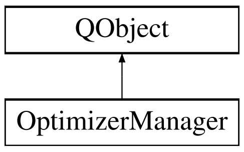
- **nfd_library** (Calibration cal_file, GPUModel &gpu_model, int x_range, int y_range, float x_inc, float y_inc)
- std::vector< nfd_instance > **get_library** ()
- void **create_nfd_library** ()

The documentation for this class was generated from the following files:

- include/nfd/nfd_library.h
- src/nfd/nfd_library.cpp

5.25 OptimizerManager Class Reference

Inheritance diagram for OptimizerManager:



Public Slots

- void **Optimize** ()
- void **onStopOptimizer** ()

Signals

- void **UpdateOptimum** (double, double, double, double, double, double, unsigned int)
- void **finished** ()
- void **OptimizedFrame** (double, double, double, double, double, double, bool, unsigned int, bool, QString)
- void **OptimizerError** (QString)
- void **UpdateDisplay** (double, int, double, unsigned int)
- void **UpdateDilationBackground** ()
- void **CostFuncAtPoint** (double)
- void **onUpdateOrientationSymTrap** (double, double, double, double, double, double)
- void **onProgressBarUpdate** (int)
- void **get_iter_count** ()

Public Member Functions

- **OptimizerManager** (QObject ∗parent=0)
- bool **Initialize** (QThread &optimizer_thread, Calibration calibration_file, std::vector< Frame > camera_↩ A_frame_list, std::vector< Frame > camera_B_frame_list, unsigned int current_frame_index, std::vector< Model > model_list, QModelIndexList selected_models, unsigned int primary_model_index, LocationStorage pose_matrix, OptimizerSettings opt_settings, jta_cost_function::CostFunctionManager trunk_manager, jta_cost_function::CostFunctionManager branch_manager, jta_cost_function::CostFunctionManager leaf_↩ manager, QString opt_directive, QString &error_message, int iter_count)
- double **EvaluateCostFunctionAtPoint** (Point6D point, int stage)
- void **CalculateSymTrap** ()

The documentation for this class was generated from the following files:

- include/core/optimizer_manager.h
- src/core/optimizer_manager.cpp

5.26 OptimizerSettings Struct Reference

Public Attributes

- Point6D **trunk_range**
- int **trunk_budget**
- Point6D **branch_range**
- int **branch_budget**
- int **number_branches**
- Point6D **leaf_range**
- int **leaf_budget**
- bool **enable_branch_**
- bool **enable_leaf_**

The documentation for this struct was generated from the following files:

- include/core/optimizer_settings.h
- src/core/optimizer_settings.cpp

5.27 jta_cost_function::Parameter< Parameter_Type > Class Template Reference

The documentation for this class was generated from the following file:

- include/cost_functions/Parameter.h

5.28 jta_cost_function::Parameter< bool > Class Reference

Public Member Functions

- **__declspec** (dllexport) Parameter()
- **__declspec** (dllexport) Parameter(std
- **__declspec** (dllexport) std
- **__declspec** (dllexport) bool getParameterValue()
- **__declspec** (dllexport) void setParameterValue(bool parameter_value)
- **__declspec** (dllexport) std

The documentation for this class was generated from the following file:

- include/cost_functions/Parameter.h

5.29 jta_cost_function::Parameter< double > Class Reference

Public Member Functions

- **__declspec** (dllexport) Parameter()
- **__declspec** (dllexport) Parameter(std
- **__declspec** (dllexport) std
- **__declspec** (dllexport) double getParameterValue()
- **__declspec** (dllexport) void setParameterValue(double parameter_value)
- **__declspec** (dllexport) std

The documentation for this class was generated from the following file:

- include/cost_functions/Parameter.h

5.30 jta_cost_function::Parameter< int > Class Reference

Public Member Functions

- **__declspec** (dllexport) Parameter()
- **__declspec** (dllexport) Parameter(std
- **__declspec** (dllexport) std
- **__declspec** (dllexport) int getParameterValue()
- **__declspec** (dllexport) void setParameterValue(int parameter_value)
- **__declspec** (dllexport) std

The documentation for this class was generated from the following file:

- include/cost_functions/Parameter.h

5.31 Point6D Struct Reference

Public Member Functions

- **Point6D** (double xval, double yval, double zval, double xaval, double yaval, double zaval)
- **Point6D** (gpu_cost_function::Pose p)
- double **GetDistanceFrom** (Point6D otherPoint)
- Direction **GetLargestDirection** ()
- double **GetDirection** (Direction direction)
- void **UpdateDirection** (Direction direction, double updated_value)

Public Attributes

- double **x**
- double **y**
- double **z**
- double **xa**
- double **ya**
- double **za**

The documentation for this struct was generated from the following files:

- include/core/data_structures_6D.h
- src/core/data_structures_6D.cpp

5.32 PoseMatrix Class Reference

Public Member Functions

- **__declspec** (dllexport) PoseMatrix()
- **__declspec** (dllexport) ∼PoseMatrix()
- __declspec(dllexport) void AddModel(std __declspec(dllexport) bool GetModelPose(std **__declspec** (dllexport) bool GetModelPose(int frame_index
- **__declspec** (dllexport) bool UpdatePrincipalModelPose(int frame_index

Public Attributes

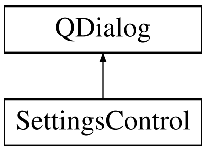
- __declspec(dllexport) void AddModel(std __declspec(dllexport) bool GetModelPose(std gpu_cost_function↩ ::Pose ∗ **pose_container**
- gpu_cost_function::Pose **pose_container**

The documentation for this class was generated from the following file:

- include/gpu/pose_matrix.h

5.33 SettingsControl Class Reference

Inheritance diagram for SettingsControl:



Public Slots

- void **on_save_button_clicked** ()
- void **on_reset_button_clicked** ()
- void **on_cancel_button_clicked** ()
- void **on_trunk_radioButton_clicked** ()
- void **on_branch_radioButton_clicked** ()
- void **on_leaf_radioButton_clicked** ()
- void **on_cost_function_listWidget_itemSelectionChanged** ()
- void **on_cost_function_parameters_listWidget_itemSelectionChanged** ()
- void **on_stage_enabled_checkBox_clicked** ()
- void **on_budget_spinBox_valueChanged** ()
- void **on_x_translation_spinBox_valueChanged** ()
- void **on_y_translation_spinBox_valueChanged** ()
- void **on_z_translation_spinBox_valueChanged** ()
- void **on_x_rotation_spinBox_valueChanged** ()
- void **on_y_rotation_spinBox_valueChanged** ()
- void **on_z_rotation_spinBox_valueChanged** ()
- void **on_branch_count_spinBox_valueChanged** ()
- void **on_double_parameter_spinBox_valueChanged** ()
- void **on_int_parameter_spinBox_valueChanged** ()
- void **on_bool_parameter_true_radioButton_clicked** ()
- void **on_bool_parameter_false_radioButton_clicked** ()

Signals

- void **SaveSettings** (OptimizerSettings, jta_cost_function::CostFunctionManager, jta_cost_function::CostFunctionManager, jta_cost_function::CostFunctionManager)
- void **Done** ()

Public Member Functions

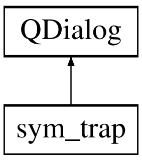
- **SettingsControl** (QWidget ∗parent=0, Qt::WindowFlags flags=0)
- void **LoadSettings** (jta_cost_function::CostFunctionManager sc_trunk_manager, jta_cost_function::CostFunctionManager sc_branch_manager, jta_cost_function::CostFunctionManager sc_leaf_manager, OptimizerSettings opt_←
settings)

The documentation for this class was generated from the following files:

- include/gui/settings_control.h
- src/gui/settings_control.cpp

5.34 sym_trap Class Reference

Inheritance diagram for sym_trap:



Public Slots

- double **onCostFuncAtPoint** (double result)
- void **graphResults** ()
- void **graphResults2D** ()
- void **setIterCount** (int n)
- void **saveData** ()
- void **loadData** ()
- void **savePlot** ()

Signals

- void **Done** ()

Public Member Functions

- **sym_trap** (QWidget ∗parent=0, Qt::WindowFlags flags=0)
- Point6D **compute_mirror_pose** (Point6D point)
- void **create_vector_of_poses** (std::vector< Point6D > &pose_list, Point6D pose)
- int **getIterCount** ()

Static Public Member Functions

- static void **matmult4** (float ans[4][4], float matrix1[4][4], float matrix2[4][4])
- static void **matmult3** (float ans[3][3], const float matrix1[3][3], const float matrix2[3][3])
- static void **invert_transform** (float result[4][4], const float tran[4][4])
- static void **equivalent_axis_angle_rotation** (float rot[3][3], const float m[3], const float angle)
- static void **cross_product** (float CP[3], const float v1[3], const float v2[3])
- static void **dot_product** (float &result, const float vector1[3], const float vector2[3])
- static void **rotation_matrix** (float R[3][3], Point6D pose)
- static void **create_312_transform** (float transform[4][4], Point6D pose)
- static void **getRotations312** (float &xr, float &yr, float &zr, const float Rot[3][3])
- static void **copy_matrix_by_value** (float(&new_matrix)[3][3], const float(&old_matrix)[3][3])
- template<typename T >
  static std::vector< double > **linspace** (T start_in, T end_in, int num_in)

Public Attributes

- Ui::symTrap **ui**

The documentation for this class was generated from the following files:

- include/gui/sym_trap.h
- src/gui/sym_trap.cpp

5.35 vec2d Struct Reference

Public Member Functions

- void **set_x** (float xval)
- void **set_y** (float yval)

Public Attributes

- float **x**
- float **y**

The documentation for this struct was generated from the following file:

- include/nfd/nfd_instance.h

5.36 Vect_3 Struct Reference

Public Member Functions

- **Vect_3** (float v_1, float v_2, float v_3)

Public Attributes

- float **v_1_**
- float **v_2_**
- float **v_3_**

The documentation for this struct was generated from the following file:

- include/core/calibration.h

5.37 viewer Class Reference

Public Member Functions

- void **initialize_vtk_pointers** ()
- void **initialize_vtk_mappers** ()
- void **load_render_window** (vtkSmartPointer< vtkRenderWindow > in)
- void **initialize_vtk_renderers** ()
- vtkSmartPointer< vtkRenderer > **get_renderer** ()
- vtkSmartPointer< vtkActor > **get_actor_image** ()
- vtkSmartPointer< vtkImageData > **get_current_background** ()
- vtkSmartPointer< vtkSTLReader > **get_stl_reader** ()
- vtkSmartPointer< vtkDataSetMapper > **get_image_mapper** ()

The documentation for this class was generated from the following files:

- include/gui/viewer.h
- src/gui/viewer.cpp

Chapter 6

File Documentation

6.1 include/core/ambiguous_pose_processing.h File Reference

This is a file that handles all of the different types of post-processing that we can do for each of the input poses. The main goal is to help with symmetry traps.

```
#include "core/sym_trap_functions.h"
#include "core/data_structures_6D.h"
```

Functions

- Point6D tibial_pose_selector (Point6D &femur_pose, Point6D &tibia_pose)

  This function will take a relative pose between a femur and tibia and choose either the current tibial pose or it's mirror.
- float varus_valgus_calculation (Point6D &femur_pose, Point6D &tibia_pose)

  This is a function that calculates the varus/valgus angle between tibia and the femur.

6.1.1 Detailed Description

This is a file that handles all of the different types of post-processing that we can do for each of the input poses. The main goal is to help with symmetry traps.

Author

Andrew Jensen (`andrewjensen321@gmail.com`)

Version

0.1

Date

2022-09-30

Copyright

Copyright (c) 2022

6.1.2 Function Documentation

6.1.2.1 tibial_pose_selector()

```
Point6D tibial_pose_selector (
            Point6D & femur_pose,
            Point6D & tibia_pose )
```

This function will take a relative pose between a femur and tibia and choose either the current tibial pose or it's mirror.

Parameters

femur_pose	This is a 6D representation of the femur pose for the current frame.
tibia_pose	This is a 6D representation of the tibial pose for the current frame.

Returns

Point6D. This function will return the (hopefully) correct tibial pose from the current pose and the mirror.

6.1.2.2 varus_valgus_calculation()

```
float varus_valgus_calculation (
            Point6D & femur_pose,
            Point6D & tibia_pose )
```

This is a function that calculates the varus/valgus angle between tibia and the femur.

Parameters

femur_pose	The femur's 6D pose.
tibia_pose	The tibia's 6D pose.

Returns

float The varus/valgus angle between the two.

6.2 ambiguous_pose_processing.h

Go to the documentation of this file.

```

#pragma once
#include "core/sym_trap_functions.h"
#include "core/data_structures_6D.h"
Point6D tibial_pose_selector(Point6D& femur_pose, Point6D& tibia_pose);

float varus_valgus_calculation(Point6D& femur_pose, Point6D& tibia_pose);
```

6.3 calibration.h

```
#ifndef CALIBRATION_H
#define CALIBRATION_H

/*Includes*/
#include "data_structures_6D.h"
#include "camera_calibration.h" //*Camera Calibration For Renderer (principal distance, principal x/y, pix
    pitch)

/*Vec 3*/
struct Vect_3 {
    Vect_3(float v_1, float v_2, float v_3) {
        v_1_ = v_1;
        v_2_ = v_2;
        v_3_ = v_3;
    }
    Vect_3() {
        v_1_ = 0;
        v_2_ = 0;
        v_3_ = 0;
    }

    /*Storage*/
    float v_1_; float v_2_; float v_3_;
};

/*3 by 3 Matrix*/
struct Matrix_3_3 {
    Matrix_3_3(float A_11, float A_12, float A_13,
        float A_21, float A_22, float A_23,
        float A_31, float A_32, float A_33) {
        A_11_ = A_11; A_12_ = A_12; A_13_ = A_13;
        A_21_ = A_21; A_22_ = A_22; A_23_ = A_23;
        A_31_ = A_31; A_32_ = A_32; A_33_ = A_33;
    };
    Matrix_3_3() {
        A_11_ = 0; A_12_ = 0; A_13_ = 0;
        A_21_ = 0; A_22_ = 0; A_23_ = 0;
        A_31_ = 0; A_32_ = 0; A_33_ = 0;
    }
    /*Storage*/
    float A_11_; float A_12_; float A_13_;
    float A_21_; float A_22_; float A_23_;
    float A_31_; float A_32_; float A_33_;

    /*Perform Transpose*/
    Matrix_3_3 tranpose() {
        return Matrix_3_3(
            A_11_, A_21_, A_31_,
            A_12_, A_22_, A_32_,
            A_13_, A_23_, A_33_);
    };
};

struct Calibration {

    /* Constructors for Monoplane and Biplane*/
    Calibration(CameraCalibration monoplane_principal) {
        biplane_calibration = false;
        camera_A_principal_ = monoplane_principal;
    };
    Calibration(CameraCalibration biplane_A_principal, CameraCalibration biplane_B_principal,
        Vect_3 origin_B, Matrix_3_3 axes_B) {
        biplane_calibration = true;
        camera_A_principal_ = biplane_A_principal;
        camera_B_principal_ = biplane_B_principal;
        origin_B_ = origin_B;
        axes_B_ = axes_B;
    };
    Calibration() {
        biplane_calibration = false;
    };

    /*Calibrated For Biplane?*/
    bool biplane_calibration;

    /*Storage*/
    CameraCalibration camera_A_principal_; /*used for both monoplane and biplane*/
    CameraCalibration camera_B_principal_; /*only used for biplane*/
    Vect_3 origin_B_; /*Origin of Camera B with respect o A which is assumed to be at (0,0,0) */
    Matrix_3_3 axes_B_; /*Orthogonal Coordinate System of B where A is assumed to have standard unit
    system*/

    /*Perform Multiplication*/
    Matrix_3_3 multiplication_mat_mat(Matrix_3_3 X, Matrix_3_3 Y) {
        return Matrix_3_3(
            X.A_11_*Y.A_11_ + X.A_12_*Y.A_21_ + X.A_13_*Y.A_31_,
            X.A_11_*Y.A_12_ + X.A_12_*Y.A_22_ + X.A_13_*Y.A_32_,
            X.A_11_*Y.A_13_ + X.A_12_*Y.A_23_ + X.A_13_*Y.A_33_,

            X.A_21_*Y.A_11_ + X.A_22_*Y.A_21_ + X.A_23_*Y.A_31_,
            X.A_21_*Y.A_12_ + X.A_22_*Y.A_22_ + X.A_23_*Y.A_32_,
            X.A_21_*Y.A_13_ + X.A_22_*Y.A_23_ + X.A_23_*Y.A_33_,

            X.A_31_*Y.A_11_ + X.A_32_*Y.A_21_ + X.A_33_*Y.A_31_,
            X.A_31_*Y.A_12_ + X.A_32_*Y.A_22_ + X.A_33_*Y.A_32_,
            X.A_31_*Y.A_13_ + X.A_32_*Y.A_23_ + X.A_33_*Y.A_33_
            );
    };
    Vect_3 multiplication_mat_vec(Matrix_3_3 X, Vect_3 u) {
        return Vect_3(
            X.A_11_*u.v_1_ + X.A_12_*u.v_2_ + X.A_13_*u.v_3_,

            X.A_21_*u.v_1_ + X.A_22_*u.v_2_ + X.A_23_*u.v_3_,

            X.A_31_*u.v_1_ + X.A_32_*u.v_2_ + X.A_33_*u.v_3_
            );
    };

    /*Camera A Pose to Camera B Pose*/
    Point6D convert_Pose_A_to_Pose_B(Point6D poseA){
        if (biplane_calibration) {
            /*Deal with Location*/
            Vect_3 location_B = multiplication_mat_vec(axes_B_.tranpose(),
                Vect_3(poseA.x - origin_B_.v_1_,
                poseA.y - origin_B_.v_2_,
                poseA.z - origin_B_.v_3_));

            /*Deal with Orientation*/
            /*Construct ROtation Matrices for A: Rz, Rx, Ry
Then Find R = Rz*Rx*Ry
Then Tranform as R_B = Q'*R where Q is the axes_B_ matrix
Then recover theta_x,y, and z for camera B (may not be unique)*/
            /*Convert To Rads*/
            float PI = 3.1415926535897932384626433832795028841;
            float theta_x_A = poseA.xa*(PI / 180.0);
            float theta_y_A = poseA.ya*(PI / 180.0);
            float theta_z_A = poseA.za*(PI / 180.0);
            Matrix_3_3 R_x(
                1, 0, 0,
                0, cos(theta_x_A), -1 * sin(theta_x_A),
                0, sin(theta_x_A), cos(theta_x_A));
            Matrix_3_3 R_y(
                cos(theta_y_A), 0, sin(theta_y_A),
                0, 1, 0,
                -1 * sin(theta_y_A), 0, cos(theta_y_A));
            Matrix_3_3 R_z(
                cos(theta_z_A), -1 * sin(theta_z_A), 0,
                sin(theta_z_A), cos(theta_z_A), 0,
                0, 0, 1);
            Matrix_3_3 R = multiplication_mat_mat(R_z, multiplication_mat_mat(R_x, R_y));
            Matrix_3_3 R_B = multiplication_mat_mat(axes_B_.tranpose(), R);

            /*Algorithm To Recover Z - X - Y Euler Angles*/
            float theta_x_B, theta_y_B, theta_z_B;
            if (R_B.A_32_ < 1) {
                if (R_B.A_32_ > -1) {
                    theta_x_B = asin(R_B.A_32_);
                    theta_z_B = atan2(-1 * R_B.A_12_, R_B.A_22_);
                    theta_y_B = atan2(-1 * R_B.A_31_, R_B.A_33_);

                }
                else {
                    theta_x_B = -PI / 2.0;
                    theta_z_B = -1 * atan2(R_B.A_13_, R_B.A_11_);
                    theta_y_B = 0;
                }
            }
            else {
                theta_x_B = PI / 2.0;
                theta_z_B = atan2(R_B.A_13_, R_B.A_11_);
                theta_y_B = 0;
            }

            /*Return New Pose*/
            return Point6D(location_B.v_1_, location_B.v_2_, location_B.v_3_,
                theta_x_B * (180.0 / PI), theta_y_B * (180.0 / PI), theta_z_B * (180.0 / PI));
        }
        else return poseA; //Just return the same.
    };

    /*Camera B Pose to Camera A Pose*/
    Point6D convert_Pose_B_to_Pose_A(Point6D poseA){
        if (biplane_calibration) {
            /*Deal with Location*/
            Vect_3 location_B = multiplication_mat_vec(axes_B_,
                Vect_3(poseA.x, poseA.y, poseA.z));
            location_B = Vect_3(location_B.v_1_ + origin_B_.v_1_,
                location_B.v_2_ + origin_B_.v_2_,
                location_B.v_3_ + origin_B_.v_3_);

            /*Deal with Orientation*/
            /*Construct ROtation Matrices for B: Rz, Rx, Ry
Then Find R = Rz*Rx*Ry
Then Tranform as R_B = Q'*R*Q where Q is the axes_B_ matrix
Then recover theta_x,y, and z for camera B (may not be unique)*/
            /*Convert To Rads*/
            float PI = 3.141592653589793238462643383279502884;
            float theta_x_A = poseA.xa*(PI / 180.0);
            float theta_y_A = poseA.ya*(PI / 180.0);
            float theta_z_A = poseA.za*(PI / 180.0);
            Matrix_3_3 R_x(
                1, 0, 0,
                0, cos(theta_x_A), -1 * sin(theta_x_A),
                0, sin(theta_x_A), cos(theta_x_A));
            Matrix_3_3 R_y(
                cos(theta_y_A), 0, sin(theta_y_A),
                0, 1, 0,
                -1 * sin(theta_y_A), 0, cos(theta_y_A));
            Matrix_3_3 R_z(
                cos(theta_z_A), -1 * sin(theta_z_A), 0,
                sin(theta_z_A), cos(theta_z_A), 0,
                0, 0, 1);
            Matrix_3_3 R = multiplication_mat_mat(R_z, multiplication_mat_mat(R_x, R_y));
            Matrix_3_3 R_B = multiplication_mat_mat(axes_B_, R);

            /*Algorithm To Recover Z - X - Y Euler Angles*/
            float theta_x_B, theta_y_B, theta_z_B;
            if (R_B.A_32_ < 1) {
                if (R_B.A_32_ > -1) {
                    theta_x_B = asin(R_B.A_32_);
                    theta_z_B = atan2(-1 * R_B.A_12_, R_B.A_22_);
                    theta_y_B = atan2(-1 * R_B.A_31_, R_B.A_33_);

                }
                else {
                    theta_x_B = -PI / 2.0;
                    theta_z_B = -1 * atan2(R_B.A_13_, R_B.A_11_);
                    theta_y_B = 0;
                }
            }
            else {
                theta_x_B = PI / 2.0;
                theta_z_B = atan2(R_B.A_13_, R_B.A_11_);
                theta_y_B = 0;
            }

            /*Return New Pose*/
            return Point6D(location_B.v_1_, location_B.v_2_, location_B.v_3_,
                theta_x_B * (180.0 / PI), theta_y_B * (180.0 / PI), theta_z_B * (180.0 / PI));
        }
        else return poseA; //Just return the same.
    };

};
#endif /* CALIBRATION_H */
```

6.4 camera_matrix.h

```
#ifndef CAMERA_MATRIX_H
#define CAMERA_MATRIX_H

#pragma once
#include "core/calibration.h"

class camera_matrix
{
public:
    camera_matrix();
    ~camera_matrix();
    camera_matrix(Calibration calibration);

private:

};

#endif
```

6.5 data_structures_6D.h

```
#ifndef DATA_STRUCTURES_6D_H
#define DATA_STRUCTURES_6D_H

/*Standard*/
#include <algorithm>
#include "gpu/render_engine.cuh"

/*Header for Data Storage Class of DIRECT algorithm (basically a linked list)*/

/*Enum Structure for Directions*/
enum Direction {
    X_DIRECTION, Y_DIRECTION, Z_DIRECTION,
    XA_DIRECTION, YA_DIRECTION, ZA_DIRECTION
};

/*Point6D to store Pose Information*/
struct Point6D
{
    Point6D(double xval, double yval, double zval, double xaval, double yaval, double zaval);
    Point6D();
    Point6D(gpu_cost_function::Pose p);

    double x; double y; double z; double xa; double ya; double za;

    double GetDistanceFrom(Point6D otherPoint);

    Direction GetLargestDirection();

    double GetDirection(Direction direction);

    void UpdateDirection(Direction direction, double updated_value);
};

/*Storage Class (Linked List of HyperMatrices/Columns) for DIRECT optimization algorithm*/
struct HyperBox6D //Stores HyperCube Info
{
    HyperBox6D(double value, Point6D center, Point6D sides);
    HyperBox6D();

    double value_;
    double size_;

    Point6D sides_;
    Point6D center_;

    void SetSides(Point6D new_sides);
    Point6D GetSides();

    bool containsPoint(Point6D point);

    Point6D GetCenter();
    void SetCenter(Point6D new_center);

    /*Divide a Side in Three*/
    void TrisectSide(Direction trisect_side);

    void PrintCenter();

};

#endif /*DATA_STRUCTURES_6D_H*/
```

6.6 direct_data_storage.h

```
#ifndef DIRECT_DATA_STORAGE_H
#define DIRECT_DATA_STORAGE_H

/*Header for Data Storage Class of DIRECT algorithm (different (faster) than one in JTA but doesnt support
    Lipschitz search)*/

/*6D Data Structures*/
#include "data_structures_6D.h"

//Standard
#include <string>
#include <vector>

class DirectDataStorage {
public:
    /*Initialize Direct Storage with a Unit Size HyperBox @ (.5, .5, .5, .5, .5, .5) with initial_value*/
    DirectDataStorage(double initial_value);
    DirectDataStorage(); /*Use Default Value of -1 For initial value*/
    //~DirectDataStorage();

    /*Remove and Add HyperBoxes*/
    void AddHyperBox(HyperBox6D *new_box);
    void DeleteHyperBoxes(std::vector<int> col_ids); /*Deletes the Best Hyperbox at the List of Column
   IDs and Deletes Empty Columns*/

    /*Get Number of Columns*/
    unsigned int GetNumberColumns();

    /*Get smallest Fvalue (last one) in column*/
    HyperBox6D GetMinimumHyperbox(int col_id);

    /*Get Smallest Fvalue (Last one) in Column*/
    double GetMinimumHyperboxValue(int col_id);

    /*Get Hyperbox Size Stored Column*/
    double GetSizeStoredInColumn(int col_id);

    /*Delete Contents of storage_matrix_ safely (also called in destructor)*/
    void DeleteAllStoredHyperboxes();

    /*Print Columns, Min/Max/Avg Column Length*/
    void PrintSize();
    /*Print */
    void PrintContents();
private:
    /*Vector of Vector of HyperBoxes:
Low Level Vector of HyperBoxes Represents All Hyperboxes of a Given Size, Kept in Sorted Decreasing (Max
   Value @ 0) Order
High Level Vector of Vectors Represents All Sizes of Current Vectors, Kept in Sorted Increasing (Min
   Value @ 0) Order*/
    std::vector<std::vector<HyperBox6D*>*> storage_matrix_;

    /*Vectors for Storing Minimum Hyperbox Size/Function Value Respectively for Each Column.
This is done for speed as it is much faster to access.  Might need to reserve space in constructor.  */
    std::vector<double> minimum_value_columns_;
    std::vector<double> size_columns_;

};

#endif /* DIRECT_DATA_STORAGE_H */
```

6.7 drr_interactor.h

```
#pragma once
/*VTK*/

#include <vtkObjectFactory.h>
#include <vtkInteractorStyleTrackballActor.h>
#include <vtkRendererCollection.h>
#include <vtkTextActor.h>
#include <vtkTextProperty.h>
#include <vtkActor2DCollection.h>
#include <vtkPicker.h>
#include <vtkPropPicker.h>
#include <vtkProp.h>
#include <qcursor.h>

//Calibration To Convert Pose
#include "core/calibration.h"

/*Drr Tool Header*/
#include "gui/drr_tool.h"

/*DRR Globals*/
bool middleDownDRR = false; // Is CM button down?
bool leftDownDRR = false; //Is LM button down?
bool rightDownDRR = false; //Is RM button down
int rightDownDRRY = 0; //Y Pixel when RM Clicked
double rightDownDRRModelZ = 0; //Model's Z Translation when RM Clicked

class DRRInteractorStyle :  public vtkInteractorStyleTrackballActor
{
public:
    static DRRInteractorStyle* New();
    vtkTypeMacro(DRRInteractorStyle, vtkInteractorStyleTrackballActor);

    /*Pointer to Main Window*/
    DRRTool* drrtool_;
    void initialize_DRRTool(DRRTool* drrtool) {
        drrtool_ = drrtool;
    }

    //Picked Function
    bool ActivePick()
    {
        if (this->InteractionProp == NULL) return false;
        else return true;
    }

    //KeyPress Turns Off Other Char Hotkeys
    virtual void OnChar() {
    }

    //Keypress Function
    virtual void OnKeyPress()
    {
        // Get the keypress
        vtkRenderWindowInteractor *rwi = this->Interactor;
        if (this->InteractionProp == NULL)
        {
            std::string key = rwi->GetKeySym();

            this->Interactor->GetRenderWindow()->Render();
            return;
        }

        vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
        std::string key = rwi->GetKeySym();
        double* Position = actor->GetPosition();

        //Shift Class
        if (rwi->GetShiftKey())
        {
            // Handle an arrow key
            if (key == "Up")
            {
                actor->RotateX(1);
                this->Interactor->GetRenderWindow()->Render();
            }
            // Handle an arrow key
            if (key == "Down")
            {
                actor->RotateX(-1 * 1);
                this->Interactor->GetRenderWindow()->Render();
            }

            // Handle an arrow key
            if (key == "Left")
            {
                actor->RotateY(-1 * 1);
                this->Interactor->GetRenderWindow()->Render();
            }

            // Handle an arrow key
            if (key == "Right")
            {
                actor->RotateY(1);
                this->Interactor->GetRenderWindow()->Render();
            }
        }
        //Control Class
        else if (rwi->GetControlKey())
         {
             // Handle an arrow key
             if (key == "Up")
             {
                 actor->SetPosition(Position[0], Position[1], Position[2] + 1);
                 this->Interactor->GetRenderWindow()->Render();
             }
             // Handle an arrow key
             if (key == "Down")
             {
                 actor->SetPosition(Position[0], Position[1], Position[2] - 1);
                 this->Interactor->GetRenderWindow()->Render();
             }

             // Handle an arrow key
             if (key == "Left")
             {
                 actor->RotateZ(-1 * 1);
                 this->Interactor->GetRenderWindow()->Render();
             }

             // Handle an arrow key
             if (key == "Right")
             {
                 actor->RotateZ(1);
                 this->Interactor->GetRenderWindow()->Render();
             }
         }
         //Naked Class
         else
         {

             // Handle an arrow key
             if (key == "Up")
             {
                 actor->SetPosition(Position[0], Position[1] + 1, Position[2]);
                 this->Interactor->GetRenderWindow()->Render();
             }
             // Handle an arrow key
             if (key == "Down")
             {
                 actor->SetPosition(Position[0], Position[1] - 1, Position[2]);
                 this->Interactor->GetRenderWindow()->Render();
             }

             // Handle an arrow key
             if (key == "Left")
             {
                 actor->SetPosition(Position[0] - 1, Position[1], Position[2]);
                 this->Interactor->GetRenderWindow()->Render();
             }

             // Handle an arrow key
             if (key == "Right")
             {
                 actor->SetPosition(Position[0] + 1, Position[1], Position[2]);
                 this->Interactor->GetRenderWindow()->Render();
             }

         }

         this->Interactor->GetRenderWindow()->Render();

         //Forward events
         vtkInteractorStyleTrackballActor::OnKeyPress();
     }

     //Left Mouse Down Function
     virtual void OnLeftButtonDown()
     {
         leftDownDRR = true;

         // Forward Events
         vtkInteractorStyleTrackballActor::OnLeftButtonDown();
     }

     //Right Mouse Down Function
     virtual void OnRightButtonDown()
     {
         rightDownDRR = true;
         rightDownDRRY = QCursor::pos().y();

         // Forward Events
         vtkInteractorStyleTrackballActor::OnRightButtonDown();

         if (this->InteractionProp == NULL)
             return;
         vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
         rightDownDRRModelZ = actor->GetPosition()[2];
     }

     //Middle Mouse Down Funtion
     virtual void OnMiddleButtonDown() {
         middleDownDRR = true;

         // Forward Events
         vtkInteractorStyleTrackballActor::OnMiddleButtonDown();
     }

     //Left Mouse Up Function
     virtual void OnLeftButtonUp()
     {
         if (this->InteractionProp == NULL)
             return;
         vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);

         leftDownDRR = false;
         this->Interactor->GetRenderWindow()->Render();
         // Forward Events
         vtkInteractorStyleTrackballActor::OnLeftButtonUp();
     }

     //Right Mouse Up Function
     virtual void OnRightButtonUp()
     {
         rightDownDRR = false;

         // Forward Events
         vtkInteractorStyleTrackballActor::OnRightButtonUp();
     }

     //Middle Mouse Up Function
     virtual void OnMiddleButtonUp() {
         middleDownDRR = false;

         //Forward Events
         vtkInteractorStyleTrackballActor::OnMiddleButtonUp();
     }

     //Mouse Movement
     virtual void OnMouseMove()
     {
         if (this->InteractionProp == NULL) return;
         if (leftDownDRR == true || rightDownDRR == true || middleDownDRR == true)
         {
             vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);

             //If Right Down and Not Left or MiddleScale The Z
             if (!leftDownDRR && !middleDownDRR)
             {
                 double* Position = actor->GetPosition();
                 actor->SetPosition(Position[0], Position[1], QCursor::pos().y() - rightDownDRRY +
     rightDownDRRModelZ);
             }
             this->Interactor->GetRenderWindow()->Render();
         }

         // Forward Events
         if (!rightDownDRR)
             vtkInteractorStyleTrackballActor::OnMouseMove();

         /*Draw DRR*/
         drrtool_->DrawDRR();
     }
 };
 vtkStandardNewMacro(DRRInteractorStyle);
```

6.8 frame.h

```
/*Header for Frame Class:
The Frame class stores:
1.  Width and Height information for an x-ray image
2.  Original x-ray image
3.  Inverted grayscale of the x-ray image
3.  Edge Detection constants (aperture, low/high threshold)
4.  Edge Detected version of the image
5.  maximum dilation constant
6.  dilated version of the edge image*/

#ifndef FRAME_H
#define FRAME_H

/*Standard*/
#include <string>

/*OpenCV 3.1 Library*/
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

class Frame {
public:
    Frame(std::string file_location, int aperture, int low_threshold,
        int high_threshold, int dilation);
    ~Frame(){};

    /*Get Set Methods for Main Variables*/
    /*Return Original Image*/
    cv::Mat GetOriginalImage();
    /*Return Edge Detected Image*/
    cv::Mat GetEdgeImage();
    /*Return Dilated Edge Detected Image*/
    cv::Mat GetDilationImage();
    /*Return Inverted Intensity Image*/
    cv::Mat GetInvertedImage();

    /*Store Public String Location*/
    std::string file_location_;

    /*Reset From Original (Resets Inverted/Segmented, Edge, Dilation from Original,
Useful if Trying to Reset from Segmentation)*/
    void ResetFromOriginal();

    /*Recalculate Edge Detected Image*/
    void SetEdgeImage(int aperture, int low_threshold,
        int high_threshold, bool use_reverse = false);
    /*Recalculate Dilated Image*/
    void SetDilatedImage(int dilation);

    /*Get Canny Parameters*/
    int GetAperture();
    int GetHighThreshold();
    int GetLowThreshold();
private:
    /*Original Matrix*/
    cv::Mat original_image_;
    /*Edge Detected Matrix*/
    cv::Mat edge_image_;
    /*Dilation Matrix*/
    cv::Mat dilation_image_;
    /*Inverted Matrix  (Store in Inverted)*/
    cv::Mat inverted_image_;

    /*Constants*/
    int aperture_;
    int low_threshold_;
    int high_threshold_;
    int dilation_;
    int width_;
    int height_;

};

#endif /* FRAME_H */
```

6.9 location_storage.h

```
/*Location Storage Class is a Matrix for Different Model Poses for Each of the Different Loaded Images (Or
    Image Pairs if Optimized*/
#ifndef LOCATION_STORAGE_H
#define LOCATION_STORAGE_H

/*Standard*/
#include <vector>

/*Direct Library*/
#include "data_structures_6D.h"

class LocationStorage {
public:
    LocationStorage(){};
    ~LocationStorage(){};

    /*Add New Model to JTA-GPU So Initialize ALl Loaded Frames with
Default Pose (0,0,-.25*principal_distance / pixel_pitch,0,0,0)*/
    void LoadNewModel(double principal_distance, double pixel_pitch);
    /*Add New Frame to JTA-GPU So Initialize ALl Loaded Models with
Default Pose (0,0,-.25*principal_distance / pixel_pitch,0,0,0) for that Frame*/
    void LoadNewFrame();

    /*Acces A Pose from Matrix*/
    Point6D GetPose(int frame_index, int model_index);

    /*Store a Pose to Matrix*/
    void SavePose(int frame_index, int model_index, Point6D model_pose);

    /*Get Frame Storage Size*/
    int GetFrameCount();

    /*Get Model Storage Size (-1 if Inconsistent Sizes)*/
    int GetModelCount();
private:
    /*List of Model Location Points for Each Frame
(So The Outside Vector is the length of the loaded frames
and the inside vector is the length of the loaded models)*/
    std::vector<std::vector<Point6D» location_storage_matrix_;

    /*No Image Model Location Points
(This stores location points for models when there are no images
loaded, this is the size of the models loaded)*/
    std::vector<Point6D> no_image_location_storage_vector_;
};

#endif /* LOCATION_STORAGE_H */
```

6.10 machine_learning_tools.h

```
#pragma once

#include <torch/torch.h>
#include <torch/script.h>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <cuda.h>
#include <cuda_runtime.h>
#include <device_launch_parameters.h>

cv::Mat segment_image(const cv::Mat& orig_image, bool black_sil_used, torch::jit::Module* model,
                      unsigned int input_width, unsigned int input_height);


```

6.11 mainscreen_size_constants.h

```
/*Constants for resizing the spacings on the main screen*/

#ifndef MAINSCREEN_SIZE_CONSTANTS_H
#define MAINSCREEN_SIZE_CONSTANTS_H

/*Sizes for Main Window*/
 int MINIMUM_WIDTH = 1600;// 1900;
 int MINIMUM_HEIGHT = 918;// 900;

/*Minimum List Widget Size*/
const int MINIMUM_LIST_WIDGET_SIZE = 100;

/*Minimum Qvtk widget width*/
const int MINIMUM_QVTK_WIDGET_WIDTH = 831;

/*Border Paddings between object A and object B*/
const int GROUP_BOX_TO_BUTTON_PADDING_X = 25;
const int BUTTON_TO_BUTTON_PADDING_X = 11;
const int INSIDE_BUTTON_PADDING_X = 30;
const int INSIDE_RADIO_BUTTON_PADDING_X = 40;
const int APPLICATION_BORDER_TO_GROUP_BOX_PADDING_X = 55;
const int INSIDE_SPIN_BOX_PADDING_X = 25;
const int LABEL_TO_SPIN_BOX_PADDING_X = 15;
const int SPIN_BOX_TO_GROUP_BOX_PADDING_X = 60;
const int INSIDE_BUTTON_PADDING_RIGHT_COLUMN_X = 50;
const int GROUP_BOX_TO_QVTK_PADDING_X = 65;

const int GROUP_BOX_TO_GROUP_BOX_Y = 30;
const int GROUP_BOX_TO_BUTTON_PADDING_Y = 30;
const int BUTTON_TO_BUTTON_PADDING_Y = 11;
const int SPIN_BOX_TO_SPIN_BOX_PADDING_Y = 15;
const int INSIDE_BUTTON_PADDING_Y = 30;
const int INSIDE_RADIO_BUTTON_PADDING_Y = 10;
const int APPLICATION_BORDER_TO_GROUP_BOX_PADDING_Y = 40;
const int RADIO_BUTTON_TO_LIST_WIDGET_PADDING_Y = 25;
const int INSIDE_SPIN_BOX_PADDING_Y = 15;

/*Font Size*/
const int FONT_SIZE = 8;

#endif /* MAINSCREEN_SIZE_CONSTANTS_H */
```

6.12 metric_enum.h

```
#ifndef METRIC_ENUM_H
#define METRIC_ENUM_H
/*Enumerator For Search Stage Flag*/
enum SearchStageFlag { Trunk = 0, Branch = 1, Leaf = 2 };

#endif /* METRIC_ENUM_H */
```

6.13 model.h

```
/*Header for Model Class Includes:
*/

/*Standard*/
#include <string>
#include <vector>

/*VTK*/
#include <vtkSmartPointer.h>
#include <vtkSTLReader.h>

/*Custom STL Reader*/
#include "stl_reader.h"

#ifndef MODEL_H
#define MODEL_H

/*AS OF VERSION 3.3.1 SHOULD BE ABLE TO LOAD BOTH BINARY AND ASCII STL FILES*/
class Model
{
public:
    Model(std::string file_location, std::string model_name, std::string model_type);
    Model(){};
    std::string file_location_; //Store File Location for Model
    vtkSmartPointer<vtkSTLReader> cad_reader_; // Stores CAD model
    std::vector<float> triangle_vertices_; //Vector of Triangle Vertices
    std::vector<float> triangle_normals_; //Vector of Triangle Normals
     /*Model Name:  taken from prefix of file name.  If duplicates a (x) is added*/
    std::string model_name_;
    /*Model Type:  could be femur or implant or bone or type of bone, anything really...*/
    std::string model_type_;
    /*Bool indicating initialized correctly*/
    bool initialized_correctly_;
private:
    stl_reader::STL_STATUS LoadVerticesAndNormals();


};

#endif /* MODEL_H */
```

6.14 optimizer_manager.h

```
/*Manages Optimization in a Seperate QT Thread*/

#ifndef OPTIMIZER_MANAGER_H
#define OPTIMIZER_MANAGER_H

/*Custom CUDA Headers*/
#include <gpu_model.cuh>
#include <gpu_intensity_frame.cuh>
#include <gpu_edge_frame.cuh>
#include <gpu_dilated_frame.cuh>
#include <gpu_metrics.cuh>
#include "calibration.h"

/*QT Threading*/
#include <qobject.h>
#include <qthread.h>
#include <QModelIndex>

/*Frame and Model and Location Storage*/
#include "frame.h"
#include "model.h"
#include "location_storage.h"

/*Direct Library*/
#include "data_structures_6D.h"
#include "direct_data_storage.h"

/*Custom Calibration Struct (Used in CUDA GPU METRICS)*/
#include "calibration.h"

/*Optimizer Settings*/
#include "optimizer_settings.h"

/*Metric Types*/
#include "metric_enum.h"

/*Cost Function Library*/
#include "cost_functions/CostFunctionManager.h"
```

```

#include "core/sym_trap_functions.h"

using namespace gpu_cost_function;

class OptimizerManager :  public QObject
{
    Q_OBJECT

public:
    explicit OptimizerManager(QObject* parent = 0);
    /*Sets up Everything for Optimizer and Also Handles CUDA Initialization, Can Fail!*/
    bool Initialize(
        QThread& optimizer_thread,
        Calibration calibration_file,
        std::vector<Frame> camera_A_frame_list, std::vector<Frame> camera_B_frame_list, unsigned int current_frame_index,
        std::vector<Model> model_list, QModelIndexList selected_models, unsigned int primary_model_index,
        LocationStorage pose_matrix,
        OptimizerSettings opt_settings,
        jta_cost_function::CostFunctionManager trunk_manager, jta_cost_function::CostFunctionManager branch_manager, jta_cost_function::CostFunctionManager leaf_manager,
        QString opt_directive,
        QString& error_message,
        int iter_count);
    ~OptimizerManager();


    /* get cost numbers for symmetry plotting */
    double EvaluateCostFunctionAtPoint(Point6D point, int stage);
    void CalculateSymTrap();

signals:
    /*Update Blue Current Optimum*/
    void UpdateOptimum(double, double, double, double, double, double, unsigned int);
    /*Finished*/
    void finished();
    /*Finished Optimizing Frame, Send Optimum to MainScreen, The last bool indicates if should move to next frame*/
    void OptimizedFrame(double, double, double, double, double, double, bool, unsigned int, bool, QString);
    /*Uh oh There was an Error.  The string is the message*/
    void OptimizerError(QString);
    /*Update Display with Speed, Cost Function Calls, Current Minimum*/
    void UpdateDisplay(double, int, double, unsigned int);
    /*Update Dilation Background*/
    void UpdateDilationBackground();

    void CostFuncAtPoint(double);
    void onUpdateOrientationSymTrap(double, double, double, double, double, double);
    void onProgressBarUpdate(int);
    void get_iter_count();

public slots:
    /*Optimizer Biplane Single Model*/
    void Optimize();

    /*Emergency Stop*/
    void onStopOptimizer();


private:

    /*Initial Variables and Objects*/
    /*Calibration File*/
    Calibration calibration_;

    /*Optimizer Settings*/
    OptimizerSettings optimizer_settings_;

    /*SYM TRAP SETTINGS*/
    bool sym_trap_call;
    //sym_trap *sym_trap_obj;

    /*Frames*/
    std::vector<Frame> frames_A_;
    /*Camera B Frames*/
    std::vector<Frame> frames_B_;

    /*Models:  All Models, Selected Non-Primary Models, and Primary Model*/
    std::vector<Model> all_models_;
    std::vector<Model> selected_non_primary_models_;
    Model primary_model_;
    /*Indices of All Selected Models*/
    QModelIndexList selected_model_list_;
    /*Index of Primary Model*/
    unsigned int primary_model_index_;

    /*Cost Function Managers For Each Stage*/
    jta_cost_function::CostFunctionManager trunk_manager_;
    jta_cost_function::CostFunctionManager branch_manager_;
    jta_cost_function::CostFunctionManager leaf_manager_;

    /*Should we progess to next frame?*/
    bool progress_next_frame_;
    /*Should we initialize with previous frame's best guess?*/
    bool init_prev_frame_;
    /*Index For Starting Frame in Optimization*/
    unsigned int start_frame_index_;
    unsigned int end_frame_index_;
    int iter_count;

    std::vector<int> img_indices_;

    QString optimization_directive_;

    void create_image_indices(std::vector<int>& img_indices, int start, int end);

    /*Error Check*/
    cudaError_t cuda_status_;

    /*Correctly Initialized*/
    bool succesfull_initialization_;

    /*Dilation Values Based on Parameter Names (Dilation or DILATION or dilation) that are ints*/
    int trunk_dilation_val_;
    int branch_dilation_val_;
    int leaf_dilation_val_;

    /*Black Silhouette Values Based on Parameter Names (Black_Silhouette or Dark_Silhouette or
  BLACK_SILHOUETTE or DARK_SILHOUETTE or black_silhouette or dark_silhouette)*/
    bool trunk_dark_silhouette_val_;
    bool branch_dark_silhouette_val_;
    bool leaf_dark_silhouette_val_;

    /*GPU Metrics Class*/
    GPUMetrics* gpu_metrics_;

    /*CUDA Cost Function Objects (Vector of GPU Models and vector of GPU Frames - note Dilated and
  Intensity must have own vector
 for each stage because their values could change with the stage from a black silhouette bool or a
  dilation int)*/
    /*Camera A (Monoplane or Biplane)*/
    std::vector<GPUIntensityFrame*> gpu_intensity_frames_trunk_A_;
    std::vector<GPUIntensityFrame*> gpu_intensity_frames_branch_A_;
    std::vector<GPUIntensityFrame*> gpu_intensity_frames_leaf_A_;
    std::vector<GPUEdgeFrame*> gpu_edge_frames_A_;
    std::vector<GPUDilatedFrame*> gpu_dilated_frames_trunk_A_;
    std::vector<GPUDilatedFrame*> gpu_dilated_frames_branch_A_;
    std::vector<GPUDilatedFrame*> gpu_dilated_frames_leaf_A_;
    /*Camera B (Biplane only)*/
    std::vector<GPUIntensityFrame*> gpu_intensity_frames_trunk_B_;
    std::vector<GPUIntensityFrame*> gpu_intensity_frames_branch_B_;
    std::vector<GPUIntensityFrame*> gpu_intensity_frames_leaf_B_;
    std::vector<GPUEdgeFrame*> gpu_edge_frames_B_;
    std::vector<GPUDilatedFrame*> gpu_dilated_frames_trunk_B_;
    std::vector<GPUDilatedFrame*> gpu_dilated_frames_branch_B_;
    std::vector<GPUDilatedFrame*> gpu_dilated_frames_leaf_B_;

    /*Models*/
    GPUModel* gpu_principal_model_;
    std::vector<GPUModel*> gpu_non_principal_models_;

    /*Set Search Range*/
    void SetSearchRange(Point6D range);

    /*Set Search Range*/
    void SetStartingPoint(Point6D starting_point);

    /*Actual Range of Search Direction for Each Variable*/
    Point6D range_;

    /*Starting Point For Search*/
    Point6D starting_point_;

    /*Valid Search Range*/
    bool valid_range_;

    /*Budget*/
    unsigned int budget_;

    /*Data Storage*/
    DirectDataStorage data_;

    /*Potentially Optimal Column Ids (Given by Convex Hull)*/
    std::vector<int> potentially_optimal_col_ids_;

    /*Potentially Optimal Hyperboxes (Taken from potentially optimal column ids)*/
    std::vector<HyperBox6D> potentially_optimal_hyperboxes_;

    /*Convex Hull Loop of DIRECT*/
    void ConvexHull();

    /*Trisect Potentially Optimal Hypers and Sample and Add
to the storage.  Delete old ones.*/
    void TrisectPotentiallyOptimal();

    /*Evaluate Cost Function at Given Point*/
    double EvaluateCostFunction(Point6D point);

    /*Denormalize Range Point (converts Unit Point to correct values)*/
    Point6D DenormalizeRange(Point6D unit_point);

    /*Denormalize Point From Center (converts Unit Point to correct values)*/
    Point6D DenormalizeFromCenter(Point6D unit_point);

    /*Cost Function Calls*/
    unsigned int cost_function_calls_;

    /*Lowest Min Value*/
    double current_optimum_value_;

    /*Argument (Location) of Lowest Min Value*/
    Point6D current_optimum_location_;

    /*Error Ocurred*/
    bool error_occurrred_;

    /*Clock for Timing Speed*/
    /*(Milliseconds)*/
    clock_t start_clock_, update_screen_clock_;

    /*Store Post Matrix on Cost Functions*/
    PoseMatrix pose_storage_;

    /*Flag For Being in Either Trunk, Branch, or Z*/
    unsigned int search_stage_flag_;

};

#endif /* OPTIMIZER_MANAGER_H */
```

6.15 optimizer_settings.h

```
#ifndef OPTIMIZER_SETTINGS_H
#define OPTIMIZER_SETTINGS_H
/*Data Structures Used by All*/
#include "data_structures_6D.h"

/*Declare as MetaType So Can Send*/
#include <QMetaType>

/*Structure that Stores the Optimizer Settings Except for the Cost Function Information which is stored in the 3 Cost Function Managers*/
struct OptimizerSettings {
    /*Constructor Destructor*/
    OptimizerSettings();
    ~OptimizerSettings();

    /*Variables*/
    /*Trunk*/
    Point6D trunk_range;
    int trunk_budget;

    /*Branch*/
    Point6D branch_range;
    int branch_budget;
    int number_branches;

    /*Leaf*/
    Point6D leaf_range;
    int leaf_budget;

    /*Optimizer Settings Which Stages Are on (trunk always on)*/
    bool enable_branch_;
    bool enable_leaf_;
};
```

```

Q_DECLARE_METATYPE(OptimizerSettings);

#endif /*OPTIMIZER_SETTINGS_H*/
```

6.16 settings_constants.h

```
/*Constants for optimizer default settings and version*/

#ifndef SETTINGS_CONSTANTS_H
#define SETTINGS_CONSTANTS_H

/*Data Structures Used by All*/
#include "data_structures_6D.h"

/*Metric Type Enumerator*/
#include "metric_enum.h"

/*Version Numbers*/
const int VER_FIRST_NUM = 3;
const int VER_MIDDLE_NUM = 4;
const int VER_LAST_NUM = 0;

/*Variables*/
/*Trunk*/
const Point6D TRUNK_RANGE = Point6D(35,35,35,35,35,35);
const int TRUNK_BUDGET = 20000;
const int TRUNK_DILATION = 6;

/*BrancheS*/
const Point6D BRANCH_RANGE = Point6D(15,15,25,25,25,25);
const int BRANCH_BUDGET = 5000;
const int NUMBER_BRANCHES = 2;
const int BRANCH_DILATION_DECREASE = 2;

/*Z- SeaRCH*/
const Point6D Z_SEARCH_RANGE = Point6D(3,3,15,3,3,3);
const int Z_SEARCH_BUDGET = 5000;
const int Z_SEARCH_DILATION = 1;

/*Display Current Optimum During Optimization*/
const bool DISPLAY_CURRENT_OPTIMUM = true;

/*Optimizer Settings Control Window Other Stuff*/
const bool ENABLE_BRANCH = true;
const bool ENABLE_Z = true;
const bool SCALE_TRUNK = false;
const double SCALE_TRUNK_VALUE = 0.5;

/*Edge Constants Save*/
const int APERTURE = 3;
const int LOW_THRESH = 40;
const int HIGH_THRESH = 120;

/*Intensity Image Uses Black Silhouette?  (True = Black, False = White)*/
const bool BLACK_SILHOUETTE = true;

/*Edge and Intensity Weights If Combined*/
const double INTENSITY_WEIGHT = 1;
const double EDGE_WEIGHT = 1;

#endif /* SETTINGS_CONSTANTS_H */
```

6.17 settings_window_size_constants.h

```
/*Constants for resizing the spacings on the main screen*/

#ifndef SETTINGS_WINDOW_SIZE_CONSTANTS_H
#define SETTINGS_WINDOW_SIZE_CONSTANTS_H


/*Border Paddings between object A and object B*/
const int BUTTON_TO_BUTTON_PADDING_X = 35;
const int INSIDE_BUTTON_PADDING_X = 60;
const int INSIDE_RADIO_BUTTON_PADDING_X = 35;
const int APPLICATION_BORDER_TO_GROUP_BOX_PADDING_X = 55;
const int INSIDE_SPIN_BOX_PADDING_X = 25;
const int LABEL_TO_SPIN_BOX_PADDING_X = 15;
const int SPIN_BOX_TO_LABEL_PADDING_X = 25;
const int SMALL_GROUP_BOX_PADDING_X =  30;
const int GROUP_BOX_TO_SMALL_GROUP_BOX_X = 30;
const int GROUP_BOX_TO_GROUP_BOX_X = 30;
const int GROUP_BOX_TO_RADIO_BUTTON_X = 40;
const int BIG_GROUP_BOX_TO_SPIN_BOX_X = 115;

const int SMALL_GROUP_BOX_PADDING_Y = 30;
const int GROUP_BOX_TO_SMALL_GROUP_BOX_Y = 30;
const int CHECKBOX_TO_LABEL_Y = 30;
const int LABEL_TO_LABEL_PADDING_Y = 25;
const int GROUP_BOX_TO_GROUP_BOX_Y = 35;
const int SMALL_GROUP_BOX_TO_GROUP_BOX_Y = 30;
const int GROUP_BOX_TO_LABEL_PADDING_Y = 30;
const int GROUP_BOX_TO_RADIO_BUTTON_PADDING_Y = 30;
const int SPIN_BOX_TO_SPIN_BOX_PADDING_Y = 15;
const int INSIDE_BUTTON_PADDING_Y = 30;
const int INSIDE_RADIO_BUTTON_PADDING_Y = 10;
const int APPLICATION_BORDER_TO_GROUP_BOX_PADDING_Y = 30;
const int INSIDE_SPIN_BOX_PADDING_Y = 15;


#endif /* SETTINGS_WINDOW_SIZE_CONSTANTS_H */
```

6.18 stl_reader.h

```
#pragma once
/*QT Headers*/
#include <QtCore\qstring.h>
#include <QtCore\qfile.h>
#include <QtCore\qfileinfo.h>
#include <QtCore\qtextstream.h>

/*Standard Library*/
#include <vector>

namespace stl_reader {

    /*ENUM for STL file status*/
    enum STL_STATUS { STL_INVALID, STL_ASCII, STL_BINARY };

    /*Function to determine if file is a valid STL file and, if so, whether it is binary or ascii*/
    STL_STATUS getStlFileFormat(const QString &path);

    /*STL reader function (binary or ascii)
Populates two vector<floats>, one contains the triangle vertices, the other contains the triangle normals
   */
    STL_STATUS readAnySTL(const QString &path, std::vector<float> &triangleVertices, std::vector<float>
   &triangleNormals);
}
```

6.19 STLReader.h

```
#pragma once
/*QT Headers*/
#include <QtCore\qstring.h>
#include <QtCore\qfile.h>
#include <QtCore\qfileinfo.h>
#include <QtCore\qtextstream.h>

/*Standard Library*/
#include <vector>

namespace stl_reader_BIG {

    /*ENUM for STL file status*/
    enum STL_STATUS { STL_INVALID, STL_ASCII, STL_BINARY };

    /*Function to determine if file is a valid STL file and, if so, whether it is binary or ascii*/
    STL_STATUS getStlFileFormat(const QString &path);

    /*STL reader function (binary or ascii)
Populates a vector of two vector<floats>, one contains the traingle vertices, the other contains the
   triangle normals  */
    STL_STATUS readAnySTL(const QString &path, std::vector<std::vector<float>> &stl_storage);
}
```

6.20 sym_trap_functions.h

```
#pragma once

#include "core/data_structures_6D.h"
#include <cmath>
#include <vector>
#include <sstream>

Point6D compute_mirror_pose(Point6D point);
void matmult4(float ans[4][4], float matrix1[4][4], float matrix2[4][4]);
void matmult3(float ans[3][3], const float matrix1[3][3], const float matrix2[3][3]);
void invert_transform(float result[4][4], const float tran[4][4]);
void equivalent_axis_angle_rotation(float rot[3][3], const float m[3], const float angle);
void cross_product(float CP[3], const float v1[3], const float v2[3]);
void dot_product(float& result, const float vector1[3], const float vector2[3]);
void rotation_matrix(float R[3][3], Point6D pose);
void create_312_transform(float transform[4][4], Point6D pose);
void getRotations312(float& xr, float& yr, float& zr, const float Rot[3][3]);

void copy_matrix_by_value(float(&new_matrix)[3][3], const float(&old_matrix)[3][3]);
void create_vector_of_poses(std::vector<Point6D>& pose_list, Point6D pose, int numPoses);

template<typename T>
std::vector<double> static linspace(T start_in, T end_in, int num_in);


```

6.21 CostFunction.h

```
#pragma once

/*Cost Function Parameters*/
#include "Parameter.h"

/*Standard Library*/
#include <vector>
#include <string>

namespace jta_cost_function {

    class CostFunction {
    public:
        /*Constructor*/
        __declspec(dllexport) CostFunction();
        __declspec(dllexport) CostFunction(std::string cost_function_name);
        __declspec(dllexport) ~CostFunction();

        /*Add Parameter (w/ Default Value)*/
        __declspec(dllexport) void addParameter(Parameter<double> new_parameter);
        __declspec(dllexport) void addParameter(Parameter<int> new_parameter);
        __declspec(dllexport) void addParameter(Parameter<bool> new_parameter);

        /*Set Parameter Values (Bool for Success)*/
        __declspec(dllexport) bool setDoubleParameterValue(std::string parameter_name, double value);
        __declspec(dllexport) bool setIntParameterValue(std::string parameter_name, int value);
        __declspec(dllexport) bool setBoolParameterValue(std::string parameter_name, bool value);

        /*Get Parameter Values (Bool for Success)*/
        __declspec(dllexport) bool getDoubleParameterValue(std::string parameter_name, double &value);
        __declspec(dllexport) bool getIntParameterValue(std::string parameter_name, int &value);
        __declspec(dllexport) bool getBoolParameterValue(std::string parameter_name, bool &value);

        /*Get Parameters by Type Groups*/
        __declspec(dllexport) std::vector<Parameter<double>> getDoubleParameters();
        __declspec(dllexport) std::vector<Parameter<int>> getIntParameters();
        __declspec(dllexport) std::vector<Parameter<bool>> getBoolParameters();

        /*Get/Set Cost Function Name*/
        __declspec(dllexport) std::string getCostFunctionName();
        __declspec(dllexport) void setCostFunctionName(std::string cost_function_name);


    private:
        /*Containers for Parameters*/
        std::vector<Parameter<double>> double_parameters_;
        std::vector<Parameter<int>> int_parameters_;
        std::vector<Parameter<bool>> bool_parameters_;

        /*Cost Function Name*/
        std::string cost_function_name_;
    };
}

```

6.22 CostFunctionManager.h

```
#ifndef COSTFUNCTIONMANAGER_H
#define COSTFUNCTIONMANAGER_H

/*Class for Storing Cost Function Info*/
#include "CostFunction.h"

/*Cost Function Tools Library*/
#include "gpu/gpu_image.cuh"
#include "gpu/gpu_frame.cuh"
#include "gpu/gpu_dilated_frame.cuh"
#include "gpu/gpu_edge_frame.cuh"
#include "gpu/gpu_intensity_frame.cuh"
#include "gpu/gpu_model.cuh"
#include "gpu/gpu_metrics.cuh"
#include "gpu/render_engine.cuh"

/*Stage Enum*/
#include "Stage.h"

/*Standard Library*/
#include <vector>

namespace jta_cost_function {
    class CostFunctionManager {
    public:
/***************************************************************************/
/************************PUBLIC DLL FUNCTIONS BEGIN ************************/
/****************************(DO NOT EDIT)  ********************************/
/***************************************************************************/
        /*Constructor
Called once when the client initially loads and populates the list of available
cost functions.  There will be three instances, one for each stage of DIRECT.
Also sets an active cost function (default is the DIRECT_DILATION).
The parameters are all default.  To load previously saved session parameters, the constructor
for the client will call the updateCostFunctionParameterValues(...)*/
        __declspec(dllexport) CostFunctionManager(Stage stage);
        __declspec(dllexport) CostFunctionManager();
        __declspec(dllexport) ~CostFunctionManager();

        /*Set Active Cost Function*/
        __declspec(dllexport) void setActiveCostFunction(std::string cost_function_name);

        /*Update Cost Function Values from Saved Session*/
        __declspec(dllexport) bool updateCostFunctionParameterValues(std::string cost_function_name,
   std::string parameter_name, double value);
        __declspec(dllexport) bool updateCostFunctionParameterValues(std::string cost_function_name,
   std::string parameter_name, int value);
        __declspec(dllexport) bool updateCostFunctionParameterValues(std::string cost_function_name,
   std::string parameter_name, bool value);

        /*Call Initialization for Active Cost Function*/
        __declspec(dllexport) bool InitializeActiveCostFunction(std::string &error_message);

        /*Call Destructor for Active Cost Function*/
        __declspec(dllexport) bool DestructActiveCostFunction(std::string &error_message);

        /*Call Active Cost Function*/
        __declspec(dllexport) double callActiveCostFunction();

        /*Get Active Cost Function*/
        __declspec(dllexport) std::string getActiveCostFunction();


        /*Get Active Cost Function Class*/
        __declspec(dllexport) CostFunction* getActiveCostFunctionClass();

        /*Get Cost Function Class*/
        __declspec(dllexport) CostFunction* getCostFunctionClass(std::string cost_function_name);

        /*Get Vector of Cost Functions*/
        __declspec(dllexport) std::vector<CostFunction> getAvailableCostFunctions();

        /*Set Current Frame Index*/
        __declspec(dllexport) void setCurrentFrameIndex(unsigned int current_frame_index);

        /*Upload Data (Images,Poses etc.)*/
        __declspec(dllexport) void UploadData(std::vector<gpu_cost_function::GPUEdgeFrame*>*
   gpu_edge_frames_A,
        std::vector<gpu_cost_function::GPUDilatedFrame*>* gpu_dilated_frames_A,
        std::vector<gpu_cost_function::GPUIntensityFrame*>* gpu_intensity_frames_A,
        std::vector<gpu_cost_function::GPUEdgeFrame*>* gpu_edge_frames_B,
        std::vector<gpu_cost_function::GPUDilatedFrame*>* gpu_dilated_frames_B,
        std::vector<gpu_cost_function::GPUIntensityFrame*>* gpu_intensity_frames_B,
        gpu_cost_function::GPUModel* gpu_principal_model,
        std::vector<gpu_cost_function::GPUModel*>* gpu_non_principal_models,
        gpu_cost_function::GPUMetrics* gpu_metrics,
        PoseMatrix* pose_storage,
        bool biplane_mode);
/******************************************************************************/
/***************************PUBLIC DLL FUNCTIONS END **************************/
/******************************************************************************/

    private:
/******************************************************************************/
        /* *******************ESSENTIAL CLASS VARIABLES BEGIN *************************/
/******************************(DO NOT EDIT)  *********************************/
/******************************************************************************/

        /*List Cost Functions
In this function a cost function that will be loaded to the client and optimizer
must be listed by name.  The parameters should also be specified.*/
        void listCostFunctions();

        /*Vector of Cost Functions*/
        std::vector<CostFunction> available_cost_functions_;

        /*Active Cost Function*/
        std::string active_cost_function_;

/******************************************************************************/
/*********************ESSENTIAL CLASS VARIABLES END ***************************/
/******************************************************************************/


/******************************************************************************/
/*********************COST FUNCTION VARIABLES BEGIN ***************************/
/******************************(DO NOT EDIT)  *********************************/
/******************************************************************************/

        /*Stage Enum*/
        Stage stage_;

        /*Storage for GPU Metrics class*/
        gpu_cost_function::GPUMetrics* gpu_metrics_;

        /*Storage for Data (images, poses ,etc.)*/
        /*Pointer to Vector of GPU Frame Pointers*/
        /*Camera A*/
        std::vector<gpu_cost_function::GPUEdgeFrame*>* gpu_edge_frames_A_;
        std::vector<gpu_cost_function::GPUDilatedFrame*>* gpu_dilated_frames_A_;
        std::vector<gpu_cost_function::GPUIntensityFrame*>* gpu_intensity_frames_A_;
        /*Camera B*/
        std::vector<gpu_cost_function::GPUEdgeFrame*>* gpu_edge_frames_B_;
        std::vector<gpu_cost_function::GPUDilatedFrame*>* gpu_dilated_frames_B_;
        std::vector<gpu_cost_function::GPUIntensityFrame*>* gpu_intensity_frames_B_;

        /*Pointer to Vector of principal GPU Model Pointer*/
        gpu_cost_function::GPUModel* gpu_principal_model_;
        /*Pointer to Vector of non-principal GPU Model Pointers*/
        std::vector<gpu_cost_function::GPUModel*>* gpu_non_principal_models_;
        float* prin_dist_;
        /*Current Frame Index (0 based)*/
        unsigned int current_frame_index_;

        /*Pose Matrix*/
        PoseMatrix* pose_storage_;

        /*Biplane Mode?*/
        bool biplane_mode_;

/******************************************************************************/
/************************COST FUNCTION VARIABLES END***************************/
/******************************************************************************/



/******************************************************************************/
/*********************CUSTOM COST FUNCTION VARIABLES BEGIN ********************/
/******************************(DO NOT EDIT)  *********************************/
/******************************************************************************/
        /*HEADERS THAT INTERACT WITH WIZARD*/
        /*Custom Variable Headers for Cost Functions*/
#include "sym_trap_functionCustomVariables.h"
#include "DD_NEW_POLE_CONSTRAINTCustomVariables.h"
#include "DIRECT_DILATION_POLE_CONSTRAINTCustomVariables.h"
#include "DIRECT_DILATION_SAME_ZCustomVariables.h"
#include "DIRECT_DILATION_T1CustomVariables.h"
#include "DIRECT_DILATIONCustomVariables.h"
#include "DIRECT_MAHFOUZCustomVariables.h"
        /*END HEADERS THAT INTERACT WITH WIZARD*/
/******************************************************************************/
/************************CUSTOM COST FUNCTION VARIABLES END********************/
```

```
/*****************************************************************************/




/********************************* WARNING ***********************************/
/*****************************************************************************/
/*************************DO NOT EDIT FUNCTIONS BELOW ************************/
/*****************************************************************************/
        /*FUNCTIONS THAT INTERACT WITH WIZARD*/
        /*Cost Function Implementations*/
        double costFunctionsym_trap_function();
        double costFunctionDD_NEW_POLE_CONSTRAINT();
        double costFunctionDIRECT_DILATION_POLE_CONSTRAINT();
        double costFunctionDIRECT_DILATION_SAME_Z();
        double costFunctionDIRECT_DILATION_T1();
        double costFunctionDIRECT_DILATION();
        double costFunctionDIRECT_MAHFOUZ();
        /*Cost Function Initializations*/
        bool initializesym_trap_function(std::string& error_message);
        bool initializeDD_NEW_POLE_CONSTRAINT(std::string& error_message);
        bool initializeDIRECT_DILATION_POLE_CONSTRAINT(std::string& error_message);
        bool initializeDIRECT_DILATION_SAME_Z(std::string& error_message);
        bool initializeDIRECT_DILATION_T1(std::string& error_message);
        bool initializeDIRECT_DILATION(std::string& error_message);
        bool initializeDIRECT_MAHFOUZ(std::string& error_message);
        /*Cost Function Destructors*/
        bool destructsym_trap_function(std::string& error_message);
        bool destructDD_NEW_POLE_CONSTRAINT(std::string& error_message);
        bool destructDIRECT_DILATION_POLE_CONSTRAINT(std::string& error_message);
        bool destructDIRECT_DILATION_SAME_Z(std::string& error_message);
        bool destructDIRECT_DILATION_T1(std::string& error_message);
        bool destructDIRECT_DILATION(std::string& error_message);
        bool destructDIRECT_MAHFOUZ(std::string& error_message);
        /*END FUNCTIONS THAT INTERACT WITH WIZARD*/
/********************************* END WARNING *******************************/
/*****************************************************************************/
/*************************DO NOT EDIT FUNCTIONS ABOVE ************************/
/*****************************************************************************/
    };
}

#endif //COSTFUNCTIONMANAGER_H
```

6.23 DD_NEW_POLE_CONSTRAINTCustomVariables.h

```
#pragma once
/****************Headers*************/
/*Cost Function Tools Library*/
#include "gpu/gpu_image.cuh"
#include "gpu/gpu_frame.cuh"
#include "gpu/gpu_dilated_frame.cuh"
#include "gpu/gpu_edge_frame.cuh"
#include "gpu/gpu_intensity_frame.cuh"
#include "gpu/gpu_model.cuh"
#include "gpu/gpu_metrics.cuh"
#include "gpu/render_engine.cuh"
/*Stage Enum*/
#include "Stage.h"
/*Parameter Class*/
#include "Parameter.h"

/****************Begin Custom Variables*************/
double x_loc_non;
double z_loc_non;
```

6.24 DIRECT_DILATION_POLE_CONSTRAINTCustomVariables.h

```
#pragma once
/****************Headers*************/
/*Cost Function Tools Library*/
#include "gpu/gpu_image.cuh"
#include "gpu/gpu_frame.cuh"
#include "gpu/gpu_dilated_frame.cuh"
#include "gpu/gpu_edge_frame.cuh"
#include "gpu/gpu_intensity_frame.cuh"
#include "gpu/gpu_model.cuh"
#include "gpu/gpu_metrics.cuh"
#include "gpu/render_engine.cuh"
```

```
/*Stage Enum*/
#include "Stage.h"
/*Parameter Class*/
#include "Parameter.h"

/****************Begin Custom Variables*************/
/*Sum of the white pixels in the current dilation comparison image*/


```

6.25 DIRECT_DILATION_SAME_ZCustomVariables.h

```
#pragma once
/****************Headers*************/
/*Cost Function Tools Library*/
#include "gpu/gpu_image.cuh"
#include "gpu/gpu_frame.cuh"
#include "gpu/gpu_dilated_frame.cuh"
#include "gpu/gpu_edge_frame.cuh"
#include "gpu/gpu_intensity_frame.cuh"
#include "gpu/gpu_model.cuh"
#include "gpu/gpu_metrics.cuh"
#include "gpu/render_engine.cuh"
/*Stage Enum*/
#include "Stage.h"
/*Parameter Class*/
#include "Parameter.h"

/****************Begin Custom Variables*************/
/*Sum of the white pixels in the current dilation comparison image*/
int DIRECT_DILATION_SAME_Z_current_white_pix_sum_dilated_comparison_image_A_;
int DIRECT_DILATION_SAME_Z_current_white_pix_sum_dilated_comparison_image_B_;
int DIRECT_DILATION_SAME_Z_current_dilation_parameter;
double DIRECT_DILATION_SAME_Z_current_z_weight_parameter;
```

6.26 DIRECT_DILATION_T1CustomVariables.h

```
#pragma once
/****************Headers*************/
/*Cost Function Tools Library*/
#include "gpu/gpu_image.cuh"
#include "gpu/gpu_frame.cuh"
#include "gpu/gpu_dilated_frame.cuh"
#include "gpu/gpu_edge_frame.cuh"
#include "gpu/gpu_intensity_frame.cuh"
#include "gpu/gpu_model.cuh"
#include "gpu/gpu_metrics.cuh"
#include "gpu/render_engine.cuh"
/*Stage Enum*/
#include "Stage.h"
/*Parameter Class*/
#include "Parameter.h"

/****************Begin Custom Variables*************/
/*Sum of the white pixels in the current dilation comparison image*/
int DIRECT_DILATION_T1_current_white_pix_sum_dilated_comparison_image_A_;
int DIRECT_DILATION_T1_current_white_pix_sum_dilated_comparison_image_B_;
int DIRECT_DILATION_T1_current_dilation_parameter;
```

6.27 DIRECT_DILATIONCustomVariables.h

```
#pragma once
/****************Headers*************/
/*Cost Function Tools Library*/
#include "gpu/gpu_image.cuh"
#include "gpu/gpu_frame.cuh"
#include "gpu/gpu_dilated_frame.cuh"
#include "gpu/gpu_edge_frame.cuh"
#include "gpu/gpu_intensity_frame.cuh"
#include "gpu/gpu_model.cuh"
#include "gpu/gpu_metrics.cuh"
#include "gpu/render_engine.cuh"
/*Stage Enum*/
#include "Stage.h"
/*Parameter Class*/
```

```
#include "Parameter.h"

/****************Begin Custom Variables*************/
/*Sum of the white pixels in the current dilation comparison image*/
int DIRECT_DILATION_current_white_pix_sum_dilated_comparison_image_A_;
int DIRECT_DILATION_current_white_pix_sum_dilated_comparison_image_B_;
int DIRECT_DILATION_current_dilation_parameter;
```

6.28 DIRECT_MAHFOUZCustomVariables.h

```
#pragma once
/****************Headers*************/
/*Cost Function Tools Library*/
#include "gpu/gpu_image.cuh"
#include "gpu/gpu_frame.cuh"
#include "gpu/gpu_dilated_frame.cuh"
#include "gpu/gpu_edge_frame.cuh"
#include "gpu/gpu_intensity_frame.cuh"
#include "gpu/gpu_model.cuh"
#include "gpu/gpu_metrics.cuh"
#include "gpu/render_engine.cuh"
/*Stage Enum*/
#include "Stage.h"
/*Parameter Class*/
#include "Parameter.h"

/****************Begin Custom Variables*************/
```

6.29 Parameter.h

```
#ifndef PARAMETER_H
#define PARAMETER_H
/*Parameter Class Header*/
/* Info:  The cost function class contains a vector of parameter classes which represent
any parameters (must be either double, integer, or bool) that the cost function might require.
These cost function parameters are added to the parameter storage vector in the constructor of
the cost function class.  Values for the parameters can be set from the JTA client and are
saved between sessions.  Default values for each parameter must be provided in the constructor
for the cost function along with a parameter type and parameter name.*/

/*Standard Library*/
#include <string>
#include <type_traits>

/*Custom Namespace for JTA Cost Function Library (Compiling as DLL)*/
namespace jta_cost_function {

    template <typename Parameter_Type>
    class Parameter {
        static_assert((std::is_same<double, Parameter_Type>::value
            || std::is_same<int, Parameter_Type>::value
            || std::is_same<bool, Parameter_Type>::value), "Parameter type must be double, int, or
   bool!");
    };

    template <>
    class Parameter<double> {
    public:
        /*Constructors*/
        __declspec(dllexport) Parameter() {
            parameter_name_ = "Nameless Parameter";
            parameter_value_ = 0;
            parameter_type_ = "DOUBLE";
        };
        __declspec(dllexport) Parameter(std::string parameter_name, double parameter_value) {
            parameter_name_ = parameter_name;
            parameter_value_ = parameter_value;
            parameter_type_ = "DOUBLE";
        };

        /*Methods*/
        /*Get Parameter Name*/
        __declspec(dllexport) std::string getParameterName() {
            return parameter_name_;
        };

        /*Get/Set Parameter Value*/
        __declspec(dllexport) double getParameterValue() {
            return parameter_value_;
        };
        __declspec(dllexport) void setParameterValue(double parameter_value) {
            parameter_value_ = parameter_value;
        };

        /*Get Class Type*/
        __declspec(dllexport) std::string getParameterType() {
            return parameter_type_;
        };

    private:
        /*Variables*/
        /*Parameter Name*/
        std::string parameter_name_;

        /*Parameter Value*/
        double parameter_value_;

        /*Class Type*/
        std::string parameter_type_;
    };

    template <>
    class Parameter<int> {
    public:
        /*Constructors*/
        __declspec(dllexport) Parameter() {
            parameter_name_ = "Nameless Parameter";
            parameter_value_ = 0;
            parameter_type_ = "INT";
        };
        __declspec(dllexport) Parameter(std::string parameter_name, int parameter_value) {
            parameter_name_ = parameter_name;
            parameter_value_ = parameter_value;
            parameter_type_ = "INT";
        };

        /*Methods*/
        /*Get Parameter Name*/
        __declspec(dllexport) std::string getParameterName() {
            return parameter_name_;
        };

        /*Get/Set Parameter Value*/
        __declspec(dllexport) int getParameterValue() {
            return parameter_value_;
        };
        __declspec(dllexport) void setParameterValue(int parameter_value) {
            parameter_value_ = parameter_value;
        };

        /*Get Class Type*/
        __declspec(dllexport) std::string getParameterType() {
            return parameter_type_;
        };

    private:
        /*Variables*/
        /*Parameter Name*/
        std::string parameter_name_;

        /*Parameter Value*/
        int parameter_value_;

        /*Class Type*/
        std::string parameter_type_;
    };

    template <>
    class Parameter<bool> {
    public:
        /*Constructors*/
        __declspec(dllexport) Parameter() {
            parameter_name_ = "Nameless Parameter";
            parameter_value_ = 0;
            parameter_type_ = "BOOL";
        };
        __declspec(dllexport) Parameter(std::string parameter_name, bool parameter_value) {
            parameter_name_ = parameter_name;
            parameter_value_ = parameter_value;
            parameter_type_ = "BOOL";
        };

        /*Methods*/
        /*Get Parameter Name*/
        __declspec(dllexport) std::string getParameterName() {
            return parameter_name_;
        };

        /*Get/Set Parameter Value*/
        __declspec(dllexport) bool getParameterValue() {
            return parameter_value_;
        };
        __declspec(dllexport) void setParameterValue(bool parameter_value) {
            parameter_value_ = parameter_value;
        };

        /*Get Class Type*/
        __declspec(dllexport) std::string getParameterType() {
            return parameter_type_;
        };

    private:
        /*Variables*/
        /*Parameter Name*/
        std::string parameter_name_;

        /*Parameter Value*/
        bool parameter_value_;

        /*Class Type*/
        std::string parameter_type_;
    };
}

#endif //PARAMETER_H
```

6.30 Stage.h

```
#ifndef STAGE_H
#define STAGE_H

/*Enum Class for Stages*/
enum class Stage { Trunk, Branch, Leaf };

#endif /*STAGE_H*/
```

6.31 sym_trap_functionCustomVariables.h

```
#pragma once
/****************Headers*************/
/*Cost Function Tools Library*/
#include "gpu/gpu_image.cuh"
#include "gpu/gpu_frame.cuh"
#include "gpu/gpu_dilated_frame.cuh"
#include "gpu/gpu_edge_frame.cuh"
#include "gpu/gpu_intensity_frame.cuh"
#include "gpu/gpu_model.cuh"
#include "gpu/gpu_metrics.cuh"
#include "gpu/render_engine.cuh"
/*Stage Enum*/
#include "Stage.h"
/*Parameter Class*/
#include "Parameter.h"

/****************Begin Custom Variables*************/
void invert_transformation(float result[4][4], float tran[4][4])
{
    int     i, j;
    /* Upper left 3x3 of result is transpose of upper left 3x3 of tran.  */
    for (i = 0; i < 3; ++i)
        for (j = 0; j < 3; ++j)
            result[i][j] = tran[j][i];
    /* Set the values for the last column of the result */
    result[3][0] = result[3][1] = result[3][2] = 0.0;
    result[3][3] = 1.0;
    /* Initialize the values of the last column of the result.  */
    result[0][3] = result[1][3] = result[2][3] = 0.0;
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            result[i][3] -= result[i][j] * tran[j][3];
        }
    }
}

void matmult(float ans[4][4], float matrix1[4][4], float matrix2[4][4])
```

```
{
    int   i, j, k;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            ans[i][j] = 0.0;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            for (k = 0; k < 4; k++)
                ans[i][j] += matrix1[i][k] * matrix2[k][j];
}

void create_312_transform(float transform[4][4], float xt, float yt, float zt, float zr, float xr, float
   yr)
{
    float degtopi = 3.1415928/180.0;
    float zr_rad = zr * degtopi;
    float xr_rad = xr * degtopi;
    float yr_rad = yr * degtopi;

    float cx = cos(xr_rad);
    float cy = cos(yr_rad);
    float cz = cos(zr_rad);
    float sx = sin(xr_rad);
    float sy = sin(yr_rad);
    float sz = sin(zr_rad);

    transform[0][0] = cy * sx * sz - cz * sy;
    transform[0][1] = -cx * sz;
    transform[0][2] = cy * cz + sx * sy * sz;
    transform[0][3] = xt;

    transform[1][0] = -cy * cz * sx - sy * sz;
    transform[1][1] = cx * cz;
    transform[1][2] = cy * sz - cz * sx * sy;
    transform[1][3] = yt;

    transform[2][0] = cx * cy;
    transform[2][1] = sx;
    transform[2][2] = cx * sy;
    transform[2][3] = zt;

    transform[3][0] = transform[3][1] = transform[3][2] = 0.0f;
    transform[3][3] = 1.0f;
}
```

6.32 camera_calibration.h

```
#ifndef CAMERA_CALIBRATION_H
#define CAMERA_CALIBRATION_H

struct CameraCalibration {
    CameraCalibration(float principal_distance, float principal_x, float principal_y, float pixel_pitch) {
        principal_distance_ = principal_distance;
        principal_x_ = principal_x;
        principal_y_ = principal_y;
        pixel_pitch_ = pixel_pitch;
     };
     CameraCalibration() {
         principal_distance_ = 0;
         principal_x_ = 0;
         principal_y_ = 0;
         pixel_pitch_ = 0;
     }
     /*Camera Location & Calibration Locations ~ Right Hand Axis System (Positive Z Towards Oneself)*/
     float principal_distance_; /* (mm) */
     float principal_x_; /* (mm) */
     float principal_y_; /* (mm) */
     float pixel_pitch_; /* pixel size in mm (mm/pixel) */
};

#endif /* CAMERA_CALIBRATION_H */
```

6.33 cuda_launch_parameters.h

```
#ifndef CUDA_LAUNCH_PARAMETERS_H
#define CUDA_LAUNCH_PARAMETERS_H

const int threads_per_block = 256;
const int maximum_stride_size = 10000000;

#endif /*CUDA_LAUNCH_PARAMETERS_H*/
```

6.34 gpu_toolbox.h

```
#ifndef GPU_TOOLBOX_H
#define GPU_TOOLBOX_H

/*CUDA Custom Registration Namespace (Compiling as DLL)*/
namespace gpu_cost_function {
    /*This class is a toolbox for users looking to write their own
cost functions, and uses GPU computing.  Users will be provided
with several resources (stored on the GPU) and functions
(computed on the GPU):
Resources (Stored on GPU Memory During Initialization of
CostFunctionToolboxGPU Class):
- Every image uploaded to JTA
- Edge detected version of every image uploaded to JTA
- Dilated version of every image uploaded to JTA
(dilation value is same as "Dilation" int parameter
in Cost Function chosen for stage.  If such a parameter
does not exist dilation is 0 and this image is a copy
of the edge detected version).
- GPU model classes (one for each model).  The GPU model
for the primary model will be stored seperately
from the list of GPU models for the non-primary
models.
- GPU model class also includes information
about the model such as the name, file location,
model type, and STL style triangle information.
This last piece of information is stored on the GPU.
- Render Engine Class.  As of 4/11/2018 this will have
to be modified to accept device pointers for the triangle
normals and vertices.


*/
    class CostFunctionToolboxGPU {
    }
}

#endif /* GPU_TOOLBOX_H */
```

6.35 pixel_grayscale_colors.h

```
#pragma once

/*Uchar Colors*/
#define WHITE_PIXEL 255
#define BLACK_PIXEL 0
#define EDGE_PIXEL 100
#define DILATED_PIXEL 99
```

6.36 pose_matrix.h

```
#pragma once

/*Render Engine Header for Pose Class*/
#include "gpu/render_engine.cuh"

/*Standard Library*/
#include <vector>
#include <string>

/*Class for Storing and Retrieving Pose linked to a unique Frame/Model Pair*/
class PoseMatrix {
public:
    /*Blank Constructor/Destructor*/
    __declspec(dllexport) PoseMatrix();
    __declspec(dllexport) ~PoseMatrix();

    /*Add New Model to Pose Matrix*/
    __declspec(dllexport) void AddModel(std::vector<gpu_cost_function::Pose > , std::string model_name,
    bool is_principal_model);

    /*Get Model Pose (True if Successful, Else False) - Pose is Returned by Passing via reference*/
    __declspec(dllexport) bool GetModelPose(std::string model_name, int frame_index,
    gpu_cost_function::Pose* pose_container);
    /*Get Principal Model Pose*/
    __declspec(dllexport) bool GetModelPose(int frame_index, gpu_cost_function::Pose* pose_container);
    /*Update Stored Pose for Principal Model at given frame*/
    __declspec(dllexport) bool UpdatePrincipalModelPose(int frame_index, gpu_cost_function::Pose
   pose_container);

private:
    /*Principal Model Name*/
    std::string principal_model_name_;

    /*Principal Model Lookup Index*/
    int principal_model_index_;

    /*Vector of All Models (Order Implies Index)*/
    std::vector<std::string> model_names_;

    /*Vector of All Model’s Vector of Frame Poses
Size of pose_matrix_ = # of models by # of frames*/
    std::vector<std::vector<gpu_cost_function::Pose» pose_matrix_;

};
```

6.37 about.h

```
#ifndef ABOUT_H
#define ABOUT_H

#include <qdialog.h>
#include "ui_About.h"

//About JTA Popup Header
class About :  public QDialog
{
    Q_OBJECT

public:
    About(QWidget *parent = 0, Qt::WindowFlags flags = 0);
    ~About();
    void setVersion(int A, int B, int C); // Sets Version Number Label

private:
    Ui::aboutJTA ui;

};

#endif // ABOUT_H
```

6.38 controls.h

```
#ifndef CONTROLS_H
#define CONTROLS_H

#include <qdialog.h>
#include "ui_controls.h"
#include <qgraphicsscene.h>
#include <qscrollbar.h>
#include <qgraphicsview.h>
#include <QGraphicsPixmapItem>
#include <qimage.h>
#include <qevent.h>

//Controls JTA Popup Header
class Controls :  public QDialog
{
    Q_OBJECT

public:
    Controls(QWidget *parent = 0, Qt::WindowFlags flags = 0);
    ~Controls();

private:
    Ui::controls ui;
    QGraphicsScene *center_scene;
    QGraphicsView *center_graph;
    QGraphicsPixmapItem *center_item;
};

#endif // CONTROLS_H
```

6.39 drr_tool.h

```
#ifndef DRR_TOOL_H
#define DRR_TOOL_H

#include <qdialog.h>
#include "ui_drr_tool.h"

/*Standard Library*/
#include <vector>

/*VTK*/
#include <vtkInteractorStyleTrackballActor.h>
#include <vtkRenderer.h>
#include <vtkSTLReader.h>
#include <vtkSmartPointer.h>
#include <vtkRenderWindow.h>
#include <vtkActor.h>
#include <vtkPolyDataMapper.h>
#include <vtkCamera.h>
#include <vtkProperty.h>
#include <vtkAlgorithm.h>

/*Models*/
#include "core/model.h"

/*GPU Models*/
#include "gpu/gpu_model.cuh"

//About JTA Popup Header
class DRRTool :  public QDialog
{
    Q_OBJECT

public:
    DRRTool(Model model, CameraCalibration calibration, double model_z_plane, QWidget *parent = 0,
   Qt::WindowFlags flags = 0);
    ~DRRTool();

    /*Draw DRR*/
    void DrawDRR();

public slots:
    /*Threshold Changes*/
    void on_minLowerSpinBox_valueChanged();
    void on_maxLowerSpinBox_valueChanged();
    void on_minUpperSpinBox_valueChanged();
    void on_maxUpperSpinBox_valueChanged();
    void on_minSlider_valueChanged();
    void on_maxSlider_valueChanged();

private:
    Ui::drrTool ui;

    /*VTK*/
    vtkSmartPointer<vtkRenderer> renderer_;

    /*Actor and Mapper and Model (CPU and GPU*)*/
    vtkSmartPointer<vtkActor> actor_;
    vtkSmartPointer<vtkPolyDataMapper> mapper_;
    Model model_;
    gpu_cost_function::GPUModel* gpu_model_;

    /*Camera Stuff*/
    float principal_distance_;
    float pixel_pitch_;
    float principal_y_;
    CameraCalibration calibration_;

    /*Array for Storing Device Image on Host*/
    unsigned char* host_image_;

    /*QImage for Converting Host to Viewable Image*/
    QImage qt_host_image_;


};

#endif // DRR_TOOL_H
```

6.40 interactor.h

```
#ifndef INTERACTOR_H
```

```
#define INTERACTOR_H

#include <vtkObjectFactory.h>
#include <vtkInteractorStyleTrackballActor.h>
#include <vtkRendererCollection.h>
#include <vtkTextActor.h>
#include <vtkTextProperty.h>
#include <vtkActor2DCollection.h>
#include <vtkPicker.h>
#include <vtkPropPicker.h>
#include <vtkProp.h>
#include <qcursor.h>

/*Ref to QMainWindow*/
#include "gui/mainscreen.h"

//Calibration To Convert Pose
#include "core/calibration.h"
Calibration interactor_calibration;

//Speed of Movement
int speed = 1;
bool information = true;
bool interactor_camera_B = false; //Are we in Camera B?
bool middleDown = false; // Is CM button down?
bool leftDown = false; //Is LM button down?
bool rightDown = false; //Is RM button down
int rightDownY = 0; //Y Pixel when RM Clicked
double rightDownModelZ = 0; //Model's Z Translation when RM Clicked

class KeyPressInteractorStyle :  public vtkInteractorStyleTrackballActor
{
public:
    static KeyPressInteractorStyle* New();
    vtkTypeMacro(KeyPressInteractorStyle, vtkInteractorStyleTrackballActor);

    /*Pointer to Main Window*/
    MainScreen* ms_;
    void initialize_MainScreen(MainScreen* ms) {
        ms_ = ms;
    }

    //Picked Function
    bool ActivePick()
    {
        if (this->InteractionProp == NULL) return false;
        else return true;
    }

    //KeyPress Turns Off Other Char Hotkeys
    virtual void OnChar() {
        vtkRenderWindowInteractor *rwi = this->Interactor;
        std::string key = rwi->GetKeySym();
        if (key == "Escape" || key == "escape" || key == "ESC" || key == "Esc" || key == "esc")
        {
            ms_->VTKEscapeSignal();
        }
    }

    //Keypress Function
    virtual void OnKeyPress()
    {
        // Get the keypress
        vtkRenderWindowInteractor *rwi = this->Interactor;
        if (this->InteractionProp == NULL)
        {
            std::string key = rwi->GetKeySym();

            // Handle information toggle
            if (key == "i" || key == "I")
            {
                vtkTextActor*  text =
   vtkTextActor::SafeDownCast(this->Interactor->GetRenderWindow()->GetRenderers()->GetFirstRenderer()->GetActors2D()->GetLa
                if (information == true) { information = false; text->GetTextProperty()->SetOpacity(0.0);
   }
                else { information = true; text->GetTextProperty()->SetOpacity(1.0); }
            }

            this->Interactor->GetRenderWindow()->Render();
            return;
        }

        vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
        std::string key = rwi->GetKeySym();
        double* Position = actor->GetPosition();

        //Shift Class
        if (rwi->GetShiftKey())
        {
            //Handle Increase Request
            if (key == "plus")
                if (speed < 20) speed++;

            //Handle Decrease Request
            if (key == "underscore")
                if (speed > 1) speed--;

            // Handle an arrow key
            if (key == "Up")
            {
                actor->RotateX(speed);
                this->Interactor->GetRenderWindow()->Render();
            }
            // Handle an arrow key
            if (key == "Down")
            {
                actor->RotateX(-1 * speed);
                this->Interactor->GetRenderWindow()->Render();
            }

            // Handle an arrow key
            if (key == "Left")
            {
                actor->RotateY(-1 * speed);
                this->Interactor->GetRenderWindow()->Render();
            }

            // Handle an arrow key
            if (key == "Right")
            {
                actor->RotateY(speed);
                this->Interactor->GetRenderWindow()->Render();
            }
        }
        //Control Class
        else if (rwi->GetControlKey())
        {
            // Handle an arrow key
            if (key == "Up")
            {
                actor->SetPosition(Position[0], Position[1], Position[2] + speed);
                this->Interactor->GetRenderWindow()->Render();
            }
            // Handle an arrow key
            if (key == "Down")
            {
                actor->SetPosition(Position[0], Position[1], Position[2] - speed);
                this->Interactor->GetRenderWindow()->Render();
            }

            // Handle an arrow key
            if (key == "Left")
            {
                actor->RotateZ(-1 * speed);
                this->Interactor->GetRenderWindow()->Render();
            }

            // Handle an arrow key
            if (key == "Right")
            {
                actor->RotateZ(speed);
                this->Interactor->GetRenderWindow()->Render();
            }
        }
        //Naked Class
        else
        {
            //Handle Increase Request
            if (key == "equal")
                if (speed < 20) speed++;

            //Handle Decrease Request
            if (key == "minus")
                if (speed > 1) speed--;

            // Handle an arrow key
            if (key == "Up")
            {
                actor->SetPosition(Position[0], Position[1] + speed, Position[2]);
                this->Interactor->GetRenderWindow()->Render();
            }
            // Handle an arrow key
            if (key == "Down")
            {
                actor->SetPosition(Position[0], Position[1] - speed, Position[2]);
                this->Interactor->GetRenderWindow()->Render();
            }

            // Handle an arrow key
            if (key == "Left")
            {
                actor->SetPosition(Position[0] - speed, Position[1], Position[2]);
                this->Interactor->GetRenderWindow()->Render();
            }

            // Handle an arrow key
            if (key == "Right")
            {
                actor->SetPosition(Position[0] + speed, Position[1], Position[2]);
                this->Interactor->GetRenderWindow()->Render();
            }

            // Handle information toggle
            if (key == "i" || key == "I")
            {
                if (information == true) information = false;
                else information = true;
            }

            // Handle information toggle
            if (key == "p" || key == "P")
            {
                if (!ms_->currently_optimizing_) {
                    ms_->VTKMakePrincipalSignal(actor);
                    return;
                }
            }

        }

        //Information Toggle
        std::string infoText = "Location:  <";
        vtkTextActor*  text =
  vtkTextActor::SafeDownCast(this->Interactor->GetRenderWindow()->GetRenderers()->GetFirstRenderer()->GetActors2D()->GetLa
        if (information == true)
        {
            if (interactor_camera_B == false) {
                infoText += std::to_string((long double)actor->GetPosition()[0]) + ","
                    + std::to_string((long double)actor->GetPosition()[1]) + ","
                    + std::to_string((long double)actor->GetPosition()[2]) + ">\nOrientation:  <"
                    + std::to_string((long double)actor->GetOrientation()[0]) + ","
                    + std::to_string((long double)actor->GetOrientation()[1]) + ","
                    + std::to_string((long double)actor->GetOrientation()[2]) + ">\nKeyboard Speed:  " +
  std::to_string((int)speed);

            }
            else {
                Point6D current_position_B = Point6D(actor->GetPosition()[0], actor->GetPosition()[1],
  actor->GetPosition()[2],
                    actor->GetOrientation()[0], actor->GetOrientation()[1], actor->GetOrientation()[2]);
                Point6D current_position_A =
  interactor_calibration.convert_Pose_B_to_Pose_A(current_position_B);
                infoText += std::to_string((long double)current_position_A.x) + ","
                    + std::to_string((long double)current_position_A.y) + ","
                    + std::to_string((long double)current_position_A.z) + ">\nOrientation:  <"
                    + std::to_string((long double)current_position_A.xa) + ","
                    + std::to_string((long double)current_position_A.ya) + ","
                    + std::to_string((long double)current_position_A.za) + ">\nKeyboard Speed:  " +
  std::to_string((int)speed);
            }
            text->GetTextProperty()->SetOpacity(1.0);
            text->GetTextProperty()->SetColor(actor->GetProperty()->GetColor());
        }
        else
            text->GetTextProperty()->SetOpacity(0.0);
        text->SetInput(infoText.c_str());
        this->Interactor->GetRenderWindow()->Render();

        //Forward events
        vtkInteractorStyleTrackballActor::OnKeyPress();
    }

    //Left Mouse Down Function
    virtual void OnLeftButtonDown()
    {
        leftDown = true;

        // Forward Events
        vtkInteractorStyleTrackballActor::OnLeftButtonDown();
    }

    //Right Mouse Down Function
    virtual void OnRightButtonDown()
    {
        rightDown = true;
        rightDownY = QCursor::pos().y();

        // Forward Events
        vtkInteractorStyleTrackballActor::OnRightButtonDown();

        if (this->InteractionProp == NULL)
            return;
        vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
        rightDownModelZ = actor->GetPosition()[2];
    }

    //Middle Mouse Down Funtion
    virtual void OnMiddleButtonDown() {
        middleDown = true;

        // Forward Events
        vtkInteractorStyleTrackballActor::OnMiddleButtonDown();
    }

    //Left Mouse Up Function
    virtual void OnLeftButtonUp()
    {
        if (this->InteractionProp == NULL)
            return;
        vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);

        leftDown = false;
        //Information Toggle
        std::string infoText = "Location:  <";
        vtkTextActor*  text =
  vtkTextActor::SafeDownCast(this->Interactor->GetRenderWindow()->GetRenderers()->GetFirstRenderer()->GetActors2D()->GetLa
        if (information == true)
        {
            if (interactor_camera_B == false) {
                infoText += std::to_string((long double)actor->GetPosition()[0]) + ","
                    + std::to_string((long double)actor->GetPosition()[1]) + ","
                    + std::to_string((long double)actor->GetPosition()[2]) + ">\nOrientation:  <"
                    + std::to_string((long double)actor->GetOrientation()[0]) + ","
                    + std::to_string((long double)actor->GetOrientation()[1]) + ","
                    + std::to_string((long double)actor->GetOrientation()[2]) + ">\nKeyboard Speed:  " +
  std::to_string((int)speed);

            }
            else {
                Point6D current_position_B = Point6D(actor->GetPosition()[0], actor->GetPosition()[1],
  actor->GetPosition()[2],
                    actor->GetOrientation()[0], actor->GetOrientation()[1], actor->GetOrientation()[2]);
                Point6D current_position_A =
  interactor_calibration.convert_Pose_B_to_Pose_A(current_position_B);
                infoText += std::to_string((long double)current_position_A.x) + ","
                    + std::to_string((long double)current_position_A.y) + ","
                    + std::to_string((long double)current_position_A.z) + ">\nOrientation:  <"
                    + std::to_string((long double)current_position_A.xa) + ","
                    + std::to_string((long double)current_position_A.ya) + ","
                    + std::to_string((long double)current_position_A.za) + ">\nKeyboard Speed:  " +
  std::to_string((int)speed);
            }
            text->GetTextProperty()->SetOpacity(1.0);
            text->GetTextProperty()->SetColor(actor->GetProperty()->GetColor());
        }
        else
            text->GetTextProperty()->SetOpacity(0.0);
        text->SetInput(infoText.c_str());
        this->Interactor->GetRenderWindow()->Render();
        // Forward Events
        vtkInteractorStyleTrackballActor::OnLeftButtonUp();
    }

    //Right Mouse Up Function
    virtual void OnRightButtonUp()
    {
        rightDown = false;

        // Forward Events
        vtkInteractorStyleTrackballActor::OnRightButtonUp();
    }

    //Middle Mouse Up Function
    virtual void OnMiddleButtonUp() {
        middleDown = false;

        //Forward Events
        vtkInteractorStyleTrackballActor::OnMiddleButtonUp();
```

```
    }

    //Mouse Movement
    virtual void OnMouseMove()
    {
        if (this->InteractionProp == NULL) return;
        if (leftDown == true || rightDown == true || middleDown == true)
        {
            vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);

            //If Right Down and Not Left or MiddleScale The Z
            if (!leftDown && !middleDown)
            {
                double* Position = actor->GetPosition();
                actor->SetPosition(Position[0], Position[1], QCursor::pos().y() - rightDownY + rightDownModelZ);
            }

            //Information Toggle
            std::string infoText = "Location:  <";
            vtkTextActor*  text = vtkTextActor::SafeDownCast(this->Interactor->GetRenderWindow()->GetRenderers()->GetFirstRenderer()->GetActors2D()->GetLa
            if (information == true)
            {
                if (interactor_camera_B == false) {
                    infoText += std::to_string((long double)actor->GetPosition()[0]) + ","
                        + std::to_string((long double)actor->GetPosition()[1]) + ","
                        + std::to_string((long double)actor->GetPosition()[2]) + ">\nOrientation:  <"
                        + std::to_string((long double)actor->GetOrientation()[0]) + ","
                        + std::to_string((long double)actor->GetOrientation()[1]) + ","
                        + std::to_string((long double)actor->GetOrientation()[2]) + ">\nKeyboard Speed: " + std::to_string((int)speed);

                }
                else {
                    Point6D current_position_B = Point6D(actor->GetPosition()[0], actor->GetPosition()[1], actor->GetPosition()[2],
                        actor->GetOrientation()[0], actor->GetOrientation()[1], actor->GetOrientation()[2]);
                    Point6D current_position_A = interactor_calibration.convert_Pose_B_to_Pose_A(current_position_B);
                    infoText += std::to_string((long double)current_position_A.x) + ","
                        + std::to_string((long double)current_position_A.y) + ","
                        + std::to_string((long double)current_position_A.z) + ">\nOrientation:  <"
                        + std::to_string((long double)current_position_A.xa) + ","
                        + std::to_string((long double)current_position_A.ya) + ","
                        + std::to_string((long double)current_position_A.za) + ">\nKeyboard Speed:  " + std::to_string((int)speed);
                }
                text->GetTextProperty()->SetOpacity(1.0);
                text->GetTextProperty()->SetColor(actor->GetProperty()->GetColor());
            }
            else
                text->GetTextProperty()->SetOpacity(0.0);
            text->SetInput(infoText.c_str());
            this->Interactor->GetRenderWindow()->Render();
        }

        // Forward Events
        if (!rightDown)
            vtkInteractorStyleTrackballActor::OnMouseMove();
    }
};
vtkStandardNewMacro(KeyPressInteractorStyle);


class CameraInteractorStyle :  public vtkInteractorStyleTrackballCamera
{
public:
    static CameraInteractorStyle* New();
    vtkTypeMacro(CameraInteractorStyle, vtkInteractorStyleTrackballCamera);

    //KeyPress Turns Off Other Char Hotkeys
    virtual void OnChar() {}
};
vtkStandardNewMacro(CameraInteractorStyle);

#endif /* INTERACTOR_H */
```

6.41 mainscreen.h

```
#ifndef MAINSCREEN_H
#define MAINSCREEN_H
```

```

/*Relevant QT Includes*/
#include <QtWidgets/QMainWindow>
#include <qactiongroup.h>
#include "ui_mainscreen.h"
#include <memory.h>
/*Font*/
#include <qfont.h>

/*Key Event*/
#include <QKeyEvent>

#include "nfd/nfd.h"

/*Direct Data Structures*/
#include "core/data_structures_6D.h"

/*Custom Calibration Struct (Used in CUDA GPU METRICS)*/
#include "core/calibration.h"

/*VTK*/
#include <vtkRenderWindow.h>
#include <vtkProperty.h>
#include <vtkCamera.h>
#include <vtkSmartPointer.h>
#include <vtkImageData.h>
#include <vtkDataSetMapper.h>
#include <vtkActor.h>
#include <vtkRenderWindow.h>
#include <vtkRenderer.h>
#include <vtkRenderWindowInteractor.h>
#include <vtkVersion.h>
#include <vtkSTLReader.h>
#include <vtkImageImport.h>
#include <vtkPolyDataMapper.h>
#include <vtkRenderWindowInteractor.h>
#include <vtkInteractorStyleTrackballActor.h>
#include <vtkTextActor.h>
#include <vtkTextProperty.h>
#include <vtkAutoInit.h> // Added post migration to Banks' lab computer
#include <vtkInteractorStyleTrackballCamera.h> /*Alternate Camera*/

/*Frame and Model and Location Storage*/
#include "core/frame.h"
#include "core/model.h"
#include "core/location_storage.h"

/*Optimizer Settings*/
#include "core/optimizer_settings.h"

/*Optimizer Manager*/
#include "core/optimizer_manager.h"

/*Optimizer Settings Control Window*/
#include "gui/settings_control.h"

/*DRR Settings Control Window*/
#include "drr_tool.h"

/* Symmetry Trap Analysis Window*/
#include "gui/sym_trap.h"

/*Cost Function Library*/
#include "cost_functions/CostFunctionManager.h"

/*CostFunctionTools*/
#include "camera_calibration.h"

/*machine_learning_tools*/
#include "core/machine_learning_tools.h"

#include "nfd/nfd.h"

#include "gui/viewer.h"

class MainScreen :  public QMainWindow
{
    Q_OBJECT

public:
    MainScreen(QWidget* parent = 0);
    ~MainScreen();

    /*Escape Signal from VTK to stop optimizer*/
    void VTKEscapeSignal();

    /*Make Selected Actor Principal from VTK*/
    void VTKMakePrincipalSignal(vtkActor* new_principal_actor);

    /*Bool to see if currently optimizing*/
    bool currently_optimizing_;




Q_SIGNALS:
    /*Update Whether To Write TO Text Display*/
     void UpdateDisplayText(bool);
     /*Stop Optimizer*/
     void StopOptimizer();

     // [SYM TRAP] Send out optimizer time remaining
     void UpdateTimeRemaining(int);

 private:
     Ui::MainScreenClass ui;

     float start_time;

     /*GUI FUNCTIONS*/
     /*Arrange Layout (Do this in code so scales across different DPI monitors and handles weird fonts)*/
     void ArrangeMainScreenLayout(QFont application_font);

     /*Private Variables*/
     /*Original Sizes After Construction for Main Screen List Widgets, their Group Boxes and QVTK Widget*/
     int image_list_widget_starting_height_;
     int image_selection_box_starting_height_;
     int model_list_widget_starting_height_;
     int model_selection_box_starting_height_;
     int qvtk_widget_starting_height_;
     int qvtk_widget_starting_width_;

     /*Monoplane and Biplane Calibration Viewport Files*/
     Calibration calibration_file_; /*Used in monoplane and biplane*/

     /*Variables Indicating Calibration Status for Mono and Biplane*/
     bool calibrated_for_monoplane_viewport_;
     bool calibrated_for_biplane_viewport_;

     /*VTK Variables*/
     std::vector<vtkSmartPointer<vtkActor» model_actor_list;
     std::vector<vtkSmartPointer<vtkPolyDataMapper» model_mapper_list;
     vtkSmartPointer<vtkRenderer> renderer;
     vtkSmartPointer<vtkImageData> current_background;
     vtkSmartPointer<vtkSTLReader> stl_reader;
     vtkSmartPointer<vtkDataSetMapper> image_mapper;
     vtkSmartPointer<vtkActor> actor_image;
     vtkSmartPointer<vtkTextActor> actor_text;
     vtkSmartPointer<vtkImageImport> importer;
     vtkSmartPointer<vtkInteractorStyleTrackballCamera> camera_style_interactor;


     std::shared_ptr<viewer> vw = std::make_shared<viewer>();


     /*View Menu Radio Button Container*/
     QActionGroup* alignmentGroup, * alignmentGroupSegment;

     /*Frame/Model Containers*/
     std::vector<Frame> loaded_frames;
     std::vector<Frame> loaded_frames_B; /*If Biplane mode, need second group of loaded frames for camera B*/
     std::vector<Model> loaded_models;

     /*Location Storage Class*/
     LocationStorage model_locations_;

     /*Index of Previously Selected Frame/Models*/
     int previous_frame_index_;
     QModelIndexList previous_model_indices_;

     /*Save the Pose From The Last Selected Frame*/
     void SaveLastPose();

     /*Optimizer Settings That Must Be Set in Constructor and Changed on OSettings Update */
     OptimizerSettings optimizer_settings_;

     /*Copy of the Above Only Used While Optimizing to Display Output*/
     OptimizerSettings display_optimizer_settings_;

     /*Cost Function Managers (from JTA Cost Function Library) for each stage of DIRECT-JTA Optimizer*/
     jta_cost_function::CostFunctionManager trunk_manager_;
     jta_cost_function::CostFunctionManager branch_manager_;
     jta_cost_function::CostFunctionManager leaf_manager_; // For extra Z-translation usually (esp.  when
   monoplane)

     /*Function That Saves Dilation as 0 if No Trunk Manager has a Dilation Int Parameter,
 else saves all the Dilation Images for Each Frame as the Dilation Constant*/
     void UpdateDilationFrames();


     /*Optimization Function:  Packages Off The Optimization process in
 a new thread*/

     /*Launch Optimizer*/
     void LaunchOptimizer(QString directive); //Directive Says whether it is Optimize Single, From, All,
   or Each (or Sym_Trap)

     /*Optimizer Thread and Manager*/
     QThread* optimizer_thread;
     OptimizerManager *optimizer_manager;

     /*Disable and Enable MainScreen During and After Optimization*/
     void DisableAll();
     void EnableAll();

     /*Function That Loads Settings from Registry or (If First Time Loading
 Saves Default Settings*/
     void LoadSettingsBetweenSessions();

     /*Mat to Vtk*/
     void matToVTK(cv::Mat Input, vtkSmartPointer<vtkImageData> Output);

     /*Optimizer Window Control*/
     SettingsControl* settings_control;

     /*Sym Trap Window*/
     sym_trap* sym_trap_control;

     /*Calculate Viewing Angle (Accounts for Offsets)*/
     double CalculateViewingAngle(int width, int height, bool CameraA);

     /*Helper Function To Segment And Update Frames According to Model File*/
     void segmentHelperFunction(std::string pt_model_location, unsigned int input_width, unsigned int
   input_height);

     // Helper function for sym_trap to get information about the current pose
     Point6D copy_current_pose();
     bool sym_trap_running;

 public Q_SLOTS:

     // Call Optimizer Launch
     void optimizer_launch_slot();

     /*Load Buttons*/
     void on_load_calibration_button_clicked(); /*Load Calibration Clicked*/
     void on_load_image_button_clicked(); /*Load Images*/
     void on_load_model_button_clicked(); /*Load Models*/


     /*Biplane View Button (Monoplane is Biplane A, Biplans is Biplane B*/
     void on_camera_A_radio_button_clicked();
     void on_camera_B_radio_button_clicked();

     /*List Widgets*/
     void on_image_list_widget_itemSelectionChanged(); /*Image List Widget Changed*/
     void on_model_list_widget_itemSelectionChanged(); /*Model List Widget Changed*/

     /*Multiple Selection For Models Radio buttons*/
     void on_single_model_radio_button_clicked();
     void on_multiple_model_radio_button_clicked();

     /*Radio Buttons*/
     /*Image Radio Buttons*/
     void on_original_image_radio_button_clicked();
     void on_inverted_image_radio_button_clicked();
     void on_edges_image_radio_button_clicked();
     void on_dilation_image_radio_button_clicked();
     /*Model Radio Buttons*/
     void on_original_model_radio_button_clicked();
     void on_solid_model_radio_button_clicked();
     void on_transparent_model_radio_button_clicked();
     void on_wireframe_model_radio_button_clicked();

     /*Edge Buttons*/
     void on_aperture_spin_box_valueChanged();
     void on_low_threshold_slider_valueChanged();
     void on_high_threshold_slider_valueChanged();
     void on_apply_all_edge_button_clicked();
     void on_reset_edge_button_clicked();

     /*MenuBar*/
     void on_actionSave_Pose_triggered();
     void on_actionSave_Kinematics_triggered();
     void on_actionLoad_Pose_triggered();
     void on_actionLoad_Kinematics_triggered();
     void on_actionAbout_JointTrack_Auto_triggered();
     void on_actionControls_triggered();
     void on_actionStop_Optimizer_triggered();
     void on_actionOptimizer_Settings_triggered();
     void on_actionDRR_Settings_triggered();
     void on_actionReset_View_triggered();
     void on_actionReset_Normal_Up_triggered();
     void on_actionModel_Interaction_Mode_triggered();
     void on_actionCamera_Interaction_Mode_triggered();
     void on_actionSegment_FemHR_triggered();
     void on_actionSegment_TibHR_triggered();
     void on_actionReset_Remove_All_Segmentation_triggered();
     void on_actionEstimate_Femoral_Implant_s_triggered();
     void on_actionEstimate_Tibial_Implant_s_triggered();

     void on_actionNFD_Pose_Estimate_triggered();

     void on_actionCopy_Next_Pose_triggered();
     void on_actionCopy_Previous_Pose_triggered();

     void on_actionLaunch_Tool_triggered();
     void on_actionAmbiguous_Pose_Processing_triggered();
     /*Optimization Buttons*/
     void on_optimize_button_clicked();
     void on_optimize_all_button_clicked();
     void on_optimize_each_button_clicked();
     void on_optimize_from_button_clicked();
     void on_actionOptimize_Backward_triggered();

     /*OPTIMIZATION SLOTS*/
     /*Update Blue Current Optimum*/
     void onUpdateOptimum(double, double, double, double, double, double, unsigned int);
     /*Finished Optimizing Frame, Send Optimum to MainScreen*/
     void onOptimizedFrame(double, double, double, double, double, double, bool, unsigned int, bool,
   QString);
     /*Uh oh There was an Error.  String contains the message
 */
     void onOptimizerError(QString error_message);
     /*Update Display with Speed, Cost Function Calls, Current Minimum*/
     void onUpdateDisplay(double, int, double, unsigned int);
     /*Update Dilation Background if Radio Button is on Dilation and Moving Betweeen Trunks and
   Branches*/
     void onUpdateDilationBackground();
     void updateOrientationSymTrap_MS(double, double, double, double, double, double);

     /*On Optimizer Control Windows Save Setting*/
     void onSaveSettings(OptimizerSettings, jta_cost_function::CostFunctionManager,
   jta_cost_function::CostFunctionManager, jta_cost_function::CostFunctionManager);

 protected:
     void resizeEvent(QResizeEvent* event);
     void keyPressEvent(QKeyEvent* event);
 };

 #endif /* MAINSCREEN_H */
```

6.42 settings_control.h

```
#ifndef SETTINGS_CONTROL_H
#define SETTINGS_CONTROL_H

#include <qdialog.h>
#include "ui_settings_control.h"

/*Optimizer Settings Class*/
#include "core/optimizer_settings.h"

/*JTA Cost Function Class*/
#include "cost_functions/CostFunctionManager.h"

//About OptimizerSettings Popup Header
class SettingsControl :  public QDialog
{
    Q_OBJECT

public:
    SettingsControl(QWidget *parent = 0, Qt::WindowFlags flags = 0);
    ~SettingsControl();

    /*Load Optimizer Settings from Main Window*/
    void LoadSettings(jta_cost_function::CostFunctionManager sc_trunk_manager,
        jta_cost_function::CostFunctionManager sc_branch_manager,
        jta_cost_function::CostFunctionManager sc_leaf_manager,
        OptimizerSettings opt_settings);

private:
    Ui::settings_control ui;

    /*Local to Settings Control Cost Function Managers*/
    jta_cost_function::CostFunctionManager sc_trunk_manager_;
    jta_cost_function::CostFunctionManager sc_branch_manager_;
    jta_cost_function::CostFunctionManager sc_leaf_manager_; // For extra Z-translation usually (esp.
   when monoplane)

    /*Optimizer Settings for Everything but the Cost Function Stuff*/
    OptimizerSettings opt_settings_;

public slots:

/*Save Button*/
void on_save_button_clicked();

/*Reset Button*/
void on_reset_button_clicked();

/*Cancel Button*/
void on_cancel_button_clicked();

/*Radio buttons for stage*/
void on_trunk_radioButton_clicked();
void on_branch_radioButton_clicked();
void on_leaf_radioButton_clicked();

/*List Widgets Changed*/
void on_cost_function_listWidget_itemSelectionChanged();
void on_cost_function_parameters_listWidget_itemSelectionChanged();

/*Optimizer Settings Buttons Toggled*/
void on_stage_enabled_checkBox_clicked();
void on_budget_spinBox_valueChanged();
void on_x_translation_spinBox_valueChanged();
void on_y_translation_spinBox_valueChanged();
void on_z_translation_spinBox_valueChanged();
void on_x_rotation_spinBox_valueChanged();
void on_y_rotation_spinBox_valueChanged();
void on_z_rotation_spinBox_valueChanged();
void on_branch_count_spinBox_valueChanged();
void on_double_parameter_spinBox_valueChanged();
void on_int_parameter_spinBox_valueChanged();
void on_bool_parameter_true_radioButton_clicked();
void on_bool_parameter_false_radioButton_clicked();



signals:
    /*Saves the three Cost Function Manager Settings and the Optimizer Settings to:
- the registry
- their local class versions on the main window GUI*/
    void SaveSettings(OptimizerSettings,
        jta_cost_function::CostFunctionManager, jta_cost_function::CostFunctionManager,
   jta_cost_function::CostFunctionManager);//
    /*Close Window*/
    void Done();//
};

#endif /* SETTINGS_CONTROL_H */
```

6.43 sym_trap.h

```
#pragma once
#include "ui_sym_trap.h"
#include "cost_functions/CostFunctionManager.h"
#include "core/data_structures_6D.h"

#include <qdialog.h>
#include <QFile>
#include <QTextStream>
#include <QtWidgets/qfiledialog.h>
#include <QMessageBox>
```

```
//#include "optimizer_manager.h"


#include <QVTKWidget.h>

#include <vtkImageData.h>
#include <vtkTextProperty.h>
#include <vtkAxis.h>
#include <vtkAxisActor2D.h>
#include <vtkChartXY.h>
#include <vtkContextScene.h>
#include <vtkContextView.h>
#include <vtkCubeAxesActor2D.h>
#include <vtkDataSetMapper.h>
#include <vtkFloatArray.h>
#include <vtkInteractorStyleTrackball.h>
#include <vtkInteractorStyleTrackballCamera.h>
#include <vtkNamedColors.h>
#include <vtkNew.h>
#include <vtkPlotPoints.h>
#include <vtkPNGWriter.h>
#include <vtkPointData.h>
#include <vtkPoints.h>
#include <vtkPolyData.h>
#include <vtkPolyDataMapper.h>
#include <vtkProperty.h>
#include <vtkRenderer.h>
#include <vtkRenderWindow.h>
#include <vtkRenderWindowInteractor.h>
#include <vtkSimplePointsReader.h>
#include <vtkSmartPointer.h>
#include <vtkTable.h>
#include <vtkWarpScalar.h>

#include <cmath>
#include <vector>
#include <iostream>
#include <sstream>


class sym_trap :public QDialog
{
    Q_OBJECT

public:
    sym_trap(QWidget* parent = 0, Qt::WindowFlags flags = 0);
    ~sym_trap();
    //OptimizerManager* sym_trap_optimizer = new OptimizerManager();


    //Point6D pose;

    Point6D compute_mirror_pose(Point6D point);
    static void matmult4(float ans[4][4], float matrix1[4][4], float matrix2[4][4]);
    static void matmult3(float ans[3][3], const float matrix1[3][3], const float matrix2[3][3]);
    static void invert_transform(float result[4][4], const float tran[4][4]);
    static void equivalent_axis_angle_rotation(float rot[3][3], const float m[3], const float angle);
    static void cross_product(float CP[3], const float v1[3], const float v2[3]);
    static void dot_product(float& result, const float vector1[3], const float vector2[3]);
    static void rotation_matrix(float R[3][3], Point6D pose);
    static void create_312_transform(float transform[4][4], Point6D pose);
    static void getRotations312(float& xr, float& yr, float& zr, const float Rot[3][3]);

    static void copy_matrix_by_value(float(&new_matrix)[3][3], const float(&old_matrix)[3][3]);
    void create_vector_of_poses(std::vector<Point6D>& pose_list, Point6D pose);

    template<typename T>
    std::vector<double> static linspace(T start_in, T end_in, int num_in);

    int getIterCount();

    //void set_pose(Point6D desired_pose);
    Ui::symTrap ui;

public Q_SLOTS:
    double onCostFuncAtPoint(double result);
    void graphResults();
    void graphResults2D();
    void setIterCount(int n);
    void saveData();
    void loadData();
    void savePlot();

private:
    std::vector<Point6D> search_space;
    QVTKWidget* plot_widget;
    int iter_count;


signals:
    void Done();
};
```

6.44 viewer.h

```
#ifndef VIEWER_H
#define VIEWER_H

#pragma once
#include <vector>

#include <vtkRenderWindow.h>
#include <vtkProperty.h>
#include <vtkCamera.h>
#include <vtkSmartPointer.h>
#include <vtkImageData.h>
#include <vtkDataSetMapper.h>
#include <vtkActor.h>
#include <vtkRenderWindow.h>
#include <vtkRenderer.h>
#include <vtkRenderWindowInteractor.h>
#include <vtkVersion.h>
#include <vtkSTLReader.h>
#include <vtkImageImport.h>
#include <vtkPolyDataMapper.h>
#include <vtkRenderWindowInteractor.h>
#include <vtkInteractorStyleTrackballActor.h>
#include <vtkTextActor.h>
#include <vtkTextProperty.h>
#include <vtkAutoInit.h> // Added post migration to Banks' lab computer
#include <vtkInteractorStyleTrackballCamera.h> /*Alternate Camera*/
#include <iostream>

class viewer
{
public:
    viewer();
    ~viewer();

    void initialize_vtk_pointers();
    void initialize_vtk_mappers();
    void load_render_window(vtkSmartPointer<vtkRenderWindow> in);
    void initialize_vtk_renderers();
    vtkSmartPointer<vtkRenderer> get_renderer();
    vtkSmartPointer<vtkActor> get_actor_image();
    vtkSmartPointer<vtkImageData> get_current_background();
    vtkSmartPointer<vtkSTLReader> get_stl_reader();
    vtkSmartPointer<vtkDataSetMapper> get_image_mapper();


private:

    std::vector<vtkSmartPointer<vtkActor» model_actor_list_;
    std::vector<vtkSmartPointer<vtkPolyDataMapper» model_mapper_list_;
    vtkSmartPointer<vtkRenderer> background_renderer_;
    vtkSmartPointer<vtkImageData> current_background_;
    vtkSmartPointer<vtkSTLReader> stl_reader_;
    vtkSmartPointer<vtkDataSetMapper> image_mapper_;
    vtkSmartPointer<vtkActor> actor_image_;
    vtkSmartPointer<vtkTextActor> actor_text_;
    vtkSmartPointer<vtkImageImport> importer_;
    vtkSmartPointer<vtkRenderWindow> qvtk_render_window_;
    vtkSmartPointer<vtkCamera> my_image_camera_;
    vtkSmartPointer<vtkCamera> my_model_camera_;

    bool initialized_pointers_;

};

#endif
```

6.45 nfd.h

```
#pragma once

#include <iostream>
```

```
#include <QString>
#include "core/model.h"
#include "core/frame.h"
#include "core/calibration.h"

#include <gpu_model.cuh>
#include <gpu_intensity_frame.cuh>
#include <gpu_edge_frame.cuh>
#include <gpu_dilated_frame.cuh>
#include <gpu_metrics.cuh>
#include <vector>
#include <QModelIndex>
#include "cufft.h"
#include <opencv2/imgproc.hpp>

#include "nfd_instance.h"
#include "nfd_library.h"

using namespace gpu_cost_function;

class JTML_NFD {

public:
    JTML_NFD();
    ~JTML_NFD();
    bool Initialize(
        Calibration cal_file,
        std::vector<Model> model_list,
        std::vector<Frame> frames_list,
        QModelIndexList selected_models,
        unsigned int primary_model_index,
        QString error_message
    );

    void Run();

private:
    bool successful_initialization_;
    Calibration calibration_;
    std::vector<Frame> frames_;

    GPUModel* gpu_principal_model_;

    Model primary_model_;
    std::vector<Model> all_models_;


};
```

6.46 nfd_instance.h

```
#pragma once
#include<complex>
#include<vector>
#include<array>
#include "gpu/render_engine.cuh"
#include "gpu/gpu_model.cuh"
#include <opencv2/core/mat.hpp>
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "alglib/interpolation.h"

using namespace gpu_cost_function;
using namespace alglib;
/*
This class is beeing used to store a single instance of the NFD libarary data.

A single instance is hereby going to refer to a single projection geometry (meaning only a single x/y
    rotation),
along with all the other data that might be associated with that.  Multiple normalization coefficients
    are going
to be defined for any one instance (as you see in the Banks paper).

The nfd_library class is going to be used to store a library of all the instances that we create for a
    single projection,
with some extra metadata surrounding it.

The main nfd function is going to manage all the initialization of the different models, as well as
    populating each of
the respective classes with the values that come from the different image projections.

*/
```

```
struct vec2d {
    float x;
    float y;
    void set_x(float xval) {
        x = xval;
    }
    void set_y(float yval) {
        y = yval;
    }
};

class nfd_instance
{
public:

    /*Constructor takes in the pose and creates the instance - will eventually also do the projections*/
    nfd_instance(GPUModel &gpu_mod,float xt, float yt, float zt, float xr, float yr, float zr);
    ~nfd_instance();

    void print_contour_points();
    void print_raw_points();

    /*Set the pose of the instance - 312 rotation order when projected*/


private:

    std::complex<float> centroid_;
    std::vector<std::complex<float>> fourier_coefficients_[128];
    float magnitude_;
    std::vector<float> angle_;
    float rot_x_;
    float rot_y_;
    Pose instance_pose_;
    std::vector<std::vector<int>> contour_pts_;
    //std::vector<int[2]> cntr_pts_;
    real_2d_array contour_points_raw_;
    std::vector<double> y_points_resampled_;
    std::vector<double> x_points_resampled_;
    pspline2interpolant contour_spline_;
    int sz_;
    void get_contour_points(cv::Mat img);

};
```

6.47 nfd_library.h

```
#pragma once
#include <vector>
#include "nfd_instance.h"
#include "nfd.h"
#include <array>
#include "gpu/gpu_model.cuh"

/*
This NFD library is going to store a vector of each of the NFD instances at a given pose, as well as some
    metadata
about the overall scope of the library (rotation parameters and ranges, etc).
*/
class nfd_library
{
public:
    nfd_library(
        Calibration cal_file,
        GPUModel &gpu_model,
        int x_range,
        int y_range,
        float x_inc,
        float y_inc
    );
    ~nfd_library();
    std::vector<nfd_instance> get_library();

    void create_nfd_library();

private:
    int x_range_;
    int y_range_;
    float x_inc_;
    float y_inc_;
    GPUModel* gpu_mod;
    std::vector<std::array<float, 2>> rot_indices_;
    void create_rot_indices(int x_range, int y_range, float x_inc, float y_inc);

    std::vector<nfd_instance> library_;
    Calibration calibration_;

    nfd_instance testing_projection();

};

```

Index