

# Joint Track Machine Learning

Generated by Doxygen 1.9.5



<b>1 Compile Instructions for Joint Track Machine Learning</b>	<b>1</b>
1.1 External Libraries	1
1.1.1 Binary Downloads	1
1.1.2 To be Built	1
1.2 In-House Libraries	2
1.3 External Library CMake Locations	2
1.4 Extras	2
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Class Documentation</b>	<b>9</b>
5.1 About Class Reference	9
5.2 Calibration Struct Reference	9
5.2.1 Constructor & Destructor Documentation	10
5.2.1.1 Calibration()	10
5.3 camera_matrix Class Reference	10
5.4 CameraCalibration Struct Reference	10
5.5 CameraInteractorStyle Class Reference	11
5.6 Controls Class Reference	11
5.7 jta_cost_function::CostFunction Class Reference	12
5.8 jta_cost_function::CostFunctionManager Class Reference	12
5.9 gpu_cost_function::CostFunctionToolboxGPU Class Reference	13
5.10 DirectDataStorage Class Reference	13
5.11 DRRInteractorStyle Class Reference	13
5.12 DRRTool Class Reference	14
5.13 Frame Class Reference	15
5.14 HyperBox6D Struct Reference	15
5.15 HyperBoxGreaterThanSize Struct Reference	16
5.16 HyperBoxLessThanValue Struct Reference	16
5.17 JTML_NFD Class Reference	16
5.18 KeyPressInteractorStyle Class Reference	16
5.19 LocationStorage Class Reference	17
5.20 MainScreen Class Reference	18
5.20.1 Constructor & Destructor Documentation	19
5.20.1.1 MainScreen()	20
5.20.2 Member Function Documentation	20
5.20.2.1 on_actionEstimate_Tibial_Implant_s_triggered	20

5.20.2.2 on_actionNFD_Pose_Estimate_triggered . . . . .	20
5.21 Matrix_3_3 Struct Reference . . . . .	20
5.22 Model Class Reference . . . . .	21
5.23 nfd_instance Class Reference . . . . .	21
5.24 nfd_library Class Reference . . . . .	22
5.25 OptimizerManager Class Reference . . . . .	22
5.26 OptimizerSettings Struct Reference . . . . .	23
5.27 jta_cost_function::Parameter< Parameter_Type > Class Template Reference . . . . .	23
5.28 jta_cost_function::Parameter< bool > Class Reference . . . . .	24
5.29 jta_cost_function::Parameter< double > Class Reference . . . . .	24
5.30 jta_cost_function::Parameter< int > Class Reference . . . . .	24
5.31 Point6D Struct Reference . . . . .	25
5.32 PoseMatrix Class Reference . . . . .	25
5.33 SettingsControl Class Reference . . . . .	26
5.34 sym_trap Class Reference . . . . .	27
5.35 vec2d Struct Reference . . . . .	28
5.36 Vect_3 Struct Reference . . . . .	28
5.37 viewer Class Reference . . . . .	29
<b>6 File Documentation</b> . . . . .	<b>31</b>
6.1 include/core/ambiguous_pose_processing.h File Reference . . . . .	31
6.1.1 Detailed Description . . . . .	31
6.1.2 Function Documentation . . . . .	32
6.1.2.1 tibial_pose_selector() . . . . .	32
6.1.2.2 varus_valgus_calculation() . . . . .	32
6.2 ambiguous_pose_processing.h . . . . .	32
6.3 calibration.h . . . . .	33
6.4 camera_matrix.h . . . . .	35
6.5 data_structures_6D.h . . . . .	36
6.6 direct_data_storage.h . . . . .	36
6.7 drr_interactor.h . . . . .	37
6.8 frame.h . . . . .	40
6.9 location_storage.h . . . . .	41
6.10 machine_learning_tools.h . . . . .	42
6.11 mainscreen_size_constants.h . . . . .	42
6.12 metric_enum.h . . . . .	42
6.13 model.h . . . . .	43
6.14 optimizer_manager.h . . . . .	43
6.15 optimizer_settings.h . . . . .	46
6.16 settings_constants.h . . . . .	47
6.17 settings_window_size_constants.h . . . . .	47
6.18 stl_reader.h . . . . .	48

6.19 STLReader.h . . . . .	48
6.20 sym_trap_functions.h . . . . .	49
6.21 CostFunction.h . . . . .	49
6.22 CostFunctionManager.h . . . . .	50
6.23 DD_NEW_POLE_CONSTRAINTCustomVariables.h . . . . .	52
6.24 DIRECT_DILATION_POLE_CONSTRAINTCustomVariables.h . . . . .	52
6.25 DIRECT_DILATION_SAME_ZCustomVariables.h . . . . .	53
6.26 DIRECT_DILATION_T1CustomVariables.h . . . . .	53
6.27 DIRECT_DILATIONCustomVariables.h . . . . .	53
6.28 DIRECT_MAHFOUZCustomVariables.h . . . . .	54
6.29 Parameter.h . . . . .	54
6.30 Stage.h . . . . .	56
6.31 sym_trap_functionCustomVariables.h . . . . .	56
6.32 camera_calibration.h . . . . .	57
6.33 cuda_launch_parameters.h . . . . .	57
6.34 gpu_toolbox.h . . . . .	58
6.35 pixel_grayscale_colors.h . . . . .	58
6.36 pose_matrix.h . . . . .	58
6.37 about.h . . . . .	59
6.38 controls.h . . . . .	59
6.39 drr_tool.h . . . . .	60
6.40 interactor.h . . . . .	60
6.41 mainscreen.h . . . . .	65
6.42 settings_control.h . . . . .	69
6.43 sym_trap.h . . . . .	70
6.44 viewer.h . . . . .	72
6.45 nfd.h . . . . .	72
6.46 nfd_instance.h . . . . .	73
6.47 nfd_library.h . . . . .	74
<b>Index</b>	<b>77</b>



## 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 download 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](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:

Calibration . . . . .	9
camera_matrix . . . . .	10
CameraCalibration . . . . .	10
jta_cost_function::CostFunction . . . . .	12
jta_cost_function::CostFunctionManager . . . . .	12
gpu_cost_function::CostFunctionToolboxGPU . . . . .	13
DirectDataStorage . . . . .	13
Frame . . . . .	15
HyperBox6D . . . . .	15
HyperBoxGreaterThanSize . . . . .	16
HyperBoxLessThanValue . . . . .	16
JTML_NFD . . . . .	16
LocationStorage . . . . .	17
Matrix_3_3 . . . . .	20
Model . . . . .	21
nfd_instance . . . . .	21
nfd_library . . . . .	22
OptimizerSettings . . . . .	23
jta_cost_function::Parameter< Parameter_Type > . . . . .	23
jta_cost_function::Parameter< bool > . . . . .	24
jta_cost_function::Parameter< double > . . . . .	24
jta_cost_function::Parameter< int > . . . . .	24
Point6D . . . . .	25
PoseMatrix . . . . .	25
QDialog	
About . . . . .	9
Controls . . . . .	11
DRRTool . . . . .	14
SettingsControl . . . . .	26
sym_trap . . . . .	27
QMainWindow	
MainScreen . . . . .	18
QObject	
OptimizerManager . . . . .	22
vec2d . . . . .	28

Vect_3 . . . . .	28
viewer . . . . .	29
vtkInteractorStyleTrackballActor	
DRRInteractorStyle . . . . .	13
KeyPressInteractorStyle . . . . .	16
vtkInteractorStyleTrackballCamera	
CameraInteractorStyle . . . . .	11

## Chapter 3

# Class Index

### 3.1 Class List

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

About	9
Calibration	9
camera_matrix	10
CameraCalibration	10
CameraInteractorStyle	11
Controls	11
jta_cost_function::CostFunction	12
jta_cost_function::CostFunctionManager	12
gpu_cost_function::CostFunctionToolboxGPU	13
DirectDataStorage	13
DRRInteractorStyle	13
DRRTool	14
Frame	15
HyperBox6D	15
HyperBoxGreaterThanSize	16
HyperBoxLessThanValue	16
JTML_NFD	16
KeyPressInteractorStyle	16
LocationStorage	17
MainScreen	18
Matrix_3_3	20
Model	21
nfd_instance	21
nfd_library	22
OptimizerManager	22
OptimizerSettings	23
jta_cost_function::Parameter< Parameter_Type >	23
jta_cost_function::Parameter< bool >	24
jta_cost_function::Parameter< double >	24
jta_cost_function::Parameter< int >	24
Point6D	25
PoseMatrix	25
SettingsControl	26
sym_trap	27
vec2d	28
Vect_3	28
viewer	29



## Chapter 4

# File Index

### 4.1 File List

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

include/core/ambiguous_pose_processing.h	
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	31
include/core/calibration.h	33
include/core/camera_matrix.h	35
include/core/data_structures_6D.h	36
include/core/direct_data_storage.h	36
include/core/drr_interactor.h	37
include/core/frame.h	40
include/core/location_storage.h	41
include/core/machine_learning_tools.h	42
include/core/mainscreen_size_constants.h	42
include/core/metric_enum.h	42
include/core/model.h	43
include/core/optimizer_manager.h	43
include/core/optimizer_settings.h	46
include/core/settings_constants.h	47
include/core/settings_window_size_constants.h	47
include/core/stl_reader.h	48
include/core/STLReader.h	48
include/core/sym_trap_functions.h	49
include/cost_functions/CostFunction.h	49
include/cost_functions/CostFunctionManager.h	50
include/cost_functions/DD_NEW_POLE_CONSTRAINTCustomVariables.h	52
include/cost_functions/DIRECT_DILATION_POLE_CONSTRAINTCustomVariables.h	52
include/cost_functions/DIRECT_DILATION_SAME_ZCustomVariables.h	53
include/cost_functions/DIRECT_DILATION_T1CustomVariables.h	53
include/cost_functions/DIRECT_DILATIONCustomVariables.h	53
include/cost_functions/DIRECT_MAHFOUZCustomVariables.h	54
include/cost_functions/Parameter.h	54
include/cost_functions/Stage.h	56
include/cost_functions/sym_trap_functionCustomVariables.h	56
include/gpu/camera_calibration.h	57
include/gpu/cuda_launch_parameters.h	57
include/gpu/gpu_toolbox.h	58

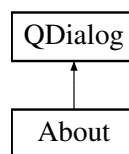
include/gpu/pixel_grayscale_colors.h . . . . .	58
include/gpu/pose_matrix.h . . . . .	58
include/gui/about.h . . . . .	59
include/gui/controls.h . . . . .	59
include/gui/drr_tool.h . . . . .	60
include/gui/interactor.h . . . . .	60
include/gui/mainscreen.h . . . . .	65
include/gui/settings_control.h . . . . .	69
include/gui/sym_trap.h . . . . .	70
include/gui/viewer.h . . . . .	72
include/nfd/nfd.h . . . . .	72
include/nfd/nfd_instance.h . . . . .	73
include/nfd/nfd_library.h . . . . .	74

## 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

<i>biplane_A_principal</i>	
<i>biplane_B_principal</i>	
<i>origin_B</i>	
<i>axes_B</i>	

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

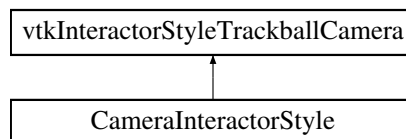
- float **principal\_distance\_**
- float **principal\_x\_**
- float **principal\_y\_**
- float **pixel\_pitch\_**

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

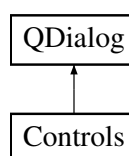
- static [CameraInteractorStyle](#) \* **New** ()

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

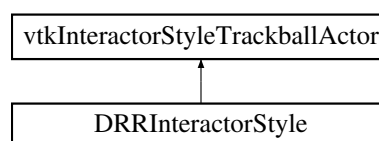
- **DirectDataStorage** (double initial\_value)
- void **AddHyperBox** ([HyperBox6D](#) \*new\_box)
- void **DeleteHyperBoxes** (std::vector< int > col\_ids)
- unsigned int **GetNumberColumns** ()
- [HyperBox6D](#) **GetMinimumHyperbox** (int col\_id)
- double **GetMinimumHyperboxValue** (int col\_id)
- 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

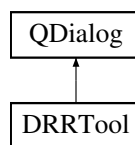
- [DRRTool](#) \* **drtool\_**

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

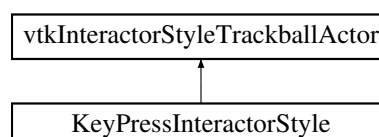
- bool **Initialize** ([Calibration](#) cal\_file, std::vector< [Model](#) > model\_list, std::vector< [Frame](#) > frames\_list, QModelIndexList selected\_models, unsigned int primary\_model\_index, QString error\_message)
- void **Run** ()

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

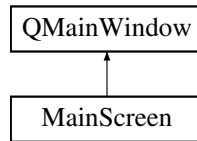
- void **LoadNewModel** (double principal\_distance, double pixel\_pitch)
- void **LoadNewFrame** ()
- [Point6D](#) **GetPose** (int frame\_index, int model\_index)
- void **SavePose** (int frame\_index, int model\_index, [Point6D](#) model\_pose)
- int **GetFrameCount** ()
- int **GetModelCount** ()

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

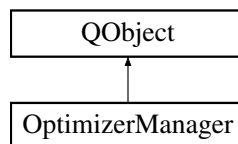
- **nfd\_library** ([Calibration](#) cal\_file, GPUModel &gpu\_model, int x\_range, int y\_range, float x\_inc, float y\_inc)
- std::vector< [nfd\\_instance](#) > **get\_library** ()
- void **create\_nfd\_library** ()

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

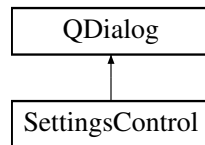
- **\_\_declspec** (dllexport) void AddModel(std **\_\_declspec** (dllexport) bool GetModelPose(std gpu\_cost\_function::Pose \* **pose\_container**
- gpu\_cost\_function::Pose **pose\_container**

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)

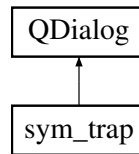
The documentation for this class was generated from the following files:

- 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](mailto: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

<i>femur_pose</i>	This is a 6D representation of the femur pose for the current frame.
<i>tibia_pose</i>	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

<i>femur_pose</i>	The femur's 6D pose.
<i>tibia_pose</i>	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.](#)

```
1
12 #pragma once
13 #include "core/sym_trap_functions.h"
14 #include "core/data_structures_6D.h"
22 Point6D tibial_pose_selector(Point6D& femur_pose, Point6D& tibia_pose);
23
31 float varus_valgus_calculation(Point6D& femur_pose, Point6D& tibia_pose);
```

## 6.3 calibration.h

```

1  #ifndef CALIBRATION_H
2  #define CALIBRATION_H
3
4  /*Includes*/
5  #include "data_structures_6D.h"
6  #include "camera_calibration.h" /*Camera Calibration For Renderer (principal distance, principal x/y, pix
    pitch)
7
8  /*Vec 3*/
9  struct Vect_3 {
10     Vect_3(float v_1, float v_2, float v_3) {
11         v_1_ = v_1;
12         v_2_ = v_2;
13         v_3_ = v_3;
14     }
15     Vect_3() {
16         v_1_ = 0;
17         v_2_ = 0;
18         v_3_ = 0;
19     }
20
21     /*Storage*/
22     float v_1_; float v_2_; float v_3_;
23 };
24
25 /*3 by 3 Matrix*/
26 struct Matrix_3_3 {
27     Matrix_3_3(float A_11, float A_12, float A_13,
28         float A_21, float A_22, float A_23,
29         float A_31, float A_32, float A_33) {
30         A_11_ = A_11; A_12_ = A_12; A_13_ = A_13;
31         A_21_ = A_21; A_22_ = A_22; A_23_ = A_23;
32         A_31_ = A_31; A_32_ = A_32; A_33_ = A_33;
33     };
34     Matrix_3_3() {
35         A_11_ = 0; A_12_ = 0; A_13_ = 0;
36         A_21_ = 0; A_22_ = 0; A_23_ = 0;
37         A_31_ = 0; A_32_ = 0; A_33_ = 0;
38     }
39     /*Storage*/
40     float A_11_; float A_12_; float A_13_;
41     float A_21_; float A_22_; float A_23_;
42     float A_31_; float A_32_; float A_33_;
43
44     /*Perform Transpose*/
45     Matrix_3_3 tranpose() {
46         return Matrix_3_3(
47             A_11_, A_21_, A_31_,
48             A_12_, A_22_, A_32_,
49             A_13_, A_23_, A_33_);
50     };
51 };
52
53 struct Calibration {
54
55     /* Constructors for Monoplane and Biplane*/
56     Calibration(CameraCalibration monoplane_principal) {
57         biplane_calibration = false;
58         camera_A_principal_ = monoplane_principal;
59     };
60     Calibration(CameraCalibration biplane_A_principal, CameraCalibration biplane_B_principal,
61         Vect_3 origin_B, Matrix_3_3 axes_B) {
62         biplane_calibration = true;
63         camera_A_principal_ = biplane_A_principal;
64         camera_B_principal_ = biplane_B_principal;
65         origin_B_ = origin_B;
66         axes_B_ = axes_B;
67     };
68     Calibration() {
69         biplane_calibration = false;
70     };
71
72     /*Calibrated For Biplane?*/
73     bool biplane_calibration;
74
75     /*Storage*/
76     CameraCalibration camera_A_principal_; /*used for both monoplane and biplane*/
77     CameraCalibration camera_B_principal_; /*only used for biplane*/
78     Vect_3 origin_B_; /*Origin of Camera B with respect o A which is assumed to be at (0,0,0) */
79     Matrix_3_3 axes_B_; /*Orthogonal Coordinate System of B where A is assumed to have standard unit
    system*/
80
81     /*Perform Multiplication*/
82     Matrix_3_3 multiplication_mat_mat(Matrix_3_3 X, Matrix_3_3 Y) {
83         return Matrix_3_3(

```

```

91         X.A_11_*Y.A_11_ + X.A_12_*Y.A_21_ + X.A_13_*Y.A_31_,
92         X.A_11_*Y.A_12_ + X.A_12_*Y.A_22_ + X.A_13_*Y.A_32_,
93         X.A_11_*Y.A_13_ + X.A_12_*Y.A_23_ + X.A_13_*Y.A_33_,
94
95         X.A_21_*Y.A_11_ + X.A_22_*Y.A_21_ + X.A_23_*Y.A_31_,
96         X.A_21_*Y.A_12_ + X.A_22_*Y.A_22_ + X.A_23_*Y.A_32_,
97         X.A_21_*Y.A_13_ + X.A_22_*Y.A_23_ + X.A_23_*Y.A_33_,
98
99         X.A_31_*Y.A_11_ + X.A_32_*Y.A_21_ + X.A_33_*Y.A_31_,
100        X.A_31_*Y.A_12_ + X.A_32_*Y.A_22_ + X.A_33_*Y.A_32_,
101        X.A_31_*Y.A_13_ + X.A_32_*Y.A_23_ + X.A_33_*Y.A_33_
102    );
103 };
104 Vect_3 multiplication_mat_vec(Matrix_3_3 X, Vect_3 u) {
105     return Vect_3(
106         X.A_11_*u.v_1_ + X.A_12_*u.v_2_ + X.A_13_*u.v_3_,
107
108         X.A_21_*u.v_1_ + X.A_22_*u.v_2_ + X.A_23_*u.v_3_,
109
110         X.A_31_*u.v_1_ + X.A_32_*u.v_2_ + X.A_33_*u.v_3_
111     );
112 };
113
114 /*Camera A Pose to Camera B Pose*/
115 Point6D convert_Pose_A_to_Pose_B(Point6D poseA){
116     if (biplane_calibration) {
117         /*Deal with Location*/
118         Vect_3 location_B = multiplication_mat_vec(axes_B_.tranpose(),
119             Vect_3(poseA.x - origin_B_.v_1_,
120                 poseA.y - origin_B_.v_2_,
121                 poseA.z - origin_B_.v_3_));
122
123         /*Deal with Orientation*/
124         /*Construct ROTation Matrices for A: Rz, Rx, Ry
125 Then Find R = Rz*Rx*Ry
126 Then Transform as R_B = Q'*R where Q is the axes_B_ matrix
127 Then recover theta_x,y, and z for camera B (may not be unique)*/
128         /*Convert To Rads*/
129         float PI = 3.141592653589793238462643383279502884;
130         float theta_x_A = poseA.xa*(PI / 180.0);
131         float theta_y_A = poseA.ya*(PI / 180.0);
132         float theta_z_A = poseA.za*(PI / 180.0);
133         Matrix_3_3 R_x(
134             1, 0, 0,
135             0, cos(theta_x_A), -1 * sin(theta_x_A),
136             0, sin(theta_x_A), cos(theta_x_A));
137         Matrix_3_3 R_y(
138             cos(theta_y_A), 0, sin(theta_y_A),
139             0, 1, 0,
140             -1 * sin(theta_y_A), 0, cos(theta_y_A));
141         Matrix_3_3 R_z(
142             cos(theta_z_A), -1 * sin(theta_z_A), 0,
143             sin(theta_z_A), cos(theta_z_A), 0,
144             0, 0, 1);
145         Matrix_3_3 R = multiplication_mat_mat(R_z, multiplication_mat_mat(R_x, R_y));
146         Matrix_3_3 R_B = multiplication_mat_mat(axes_B_.tranpose(), R);
147
148         /*Algorithm To Recover Z - X - Y Euler Angles*/
149         float theta_x_B, theta_y_B, theta_z_B;
150         if (R_B.A_32_ < 1) {
151             if (R_B.A_32_ > -1) {
152                 theta_x_B = asin(R_B.A_32_);
153                 theta_z_B = atan2(-1 * R_B.A_12_, R_B.A_22_);
154                 theta_y_B = atan2(-1 * R_B.A_31_, R_B.A_33_);
155             }
156             else {
157                 theta_x_B = -PI / 2.0;
158                 theta_z_B = -1 * atan2(R_B.A_13_, R_B.A_11_);
159                 theta_y_B = 0;
160             }
161         }
162         else {
163             theta_x_B = PI / 2.0;
164             theta_z_B = atan2(R_B.A_13_, R_B.A_11_);
165             theta_y_B = 0;
166         }
167
168         /*Return New Pose*/
169         return Point6D(location_B.v_1_, location_B.v_2_, location_B.v_3_,
170             theta_x_B * (180.0 / PI), theta_y_B * (180.0 / PI), theta_z_B * (180.0 / PI));
171     }
172     else return poseA; //Just return the same.
173 };
174
175 /*Camera B Pose to Camera A Pose*/
176 Point6D convert_Pose_B_to_Pose_A(Point6D poseA){

```



```

178         if (biplane_calibration) {
179             /*Deal with Location*/
180             Vect_3 location_B = multiplication_mat_vec(axes_B_,
181             Vect_3(poseA.x, poseA.y, poseA.z));
182             location_B = Vect_3(location_B.v_1_ + origin_B_.v_1_,
183             location_B.v_2_ + origin_B_.v_2_,
184             location_B.v_3_ + origin_B_.v_3_);
185
186             /*Deal with Orientation*/
187             /*Construct ROTation Matrices for B: Rz, Rx, Ry
188             Then Find R = Rz*Rx*Ry
189             Then Transform as R_B = Q'*R*Q where Q is the axes_B_ matrix
190             Then recover theta_x,y, and z for camera B (may not be unique)*/
191             /*Convert To Rads*/
192             float PI = 3.141592653589793238462643383279502884;
193             float theta_x_A = poseA.xa*(PI / 180.0);
194             float theta_y_A = poseA.ya*(PI / 180.0);
195             float theta_z_A = poseA.za*(PI / 180.0);
196             Matrix_3_3 R_x(
197             1, 0, 0,
198             0, cos(theta_x_A), -1 * sin(theta_x_A),
199             0, sin(theta_x_A), cos(theta_x_A));
200             Matrix_3_3 R_y(
201             cos(theta_y_A), 0, sin(theta_y_A),
202             0, 1, 0,
203             -1 * sin(theta_y_A), 0, cos(theta_y_A));
204             Matrix_3_3 R_z(
205             cos(theta_z_A), -1 * sin(theta_z_A), 0,
206             sin(theta_z_A), cos(theta_z_A), 0,
207             0, 0, 1);
208             Matrix_3_3 R = multiplication_mat_mat(R_z, multiplication_mat_mat(R_x, R_y));
209             Matrix_3_3 R_B = multiplication_mat_mat(axes_B_, R);
210
211             /*Algorithm To Recover Z - X - Y Euler Angles*/
212             float theta_x_B, theta_y_B, theta_z_B;
213             if (R_B.A_32_ < 1) {
214                 if (R_B.A_32_ > -1) {
215                     theta_x_B = asin(R_B.A_32_);
216                     theta_z_B = atan2(-1 * R_B.A_12_, R_B.A_22_);
217                     theta_y_B = atan2(-1 * R_B.A_31_, R_B.A_33_);
218                 }
219                 else {
220                     theta_x_B = -PI / 2.0;
221                     theta_z_B = -1 * atan2(R_B.A_13_, R_B.A_11_);
222                     theta_y_B = 0;
223                 }
224             }
225             else {
226                 theta_x_B = PI / 2.0;
227                 theta_z_B = atan2(R_B.A_13_, R_B.A_11_);
228                 theta_y_B = 0;
229             }
230
231             /*Return New Pose*/
232             return Point6D(location_B.v_1_, location_B.v_2_, location_B.v_3_,
233             theta_x_B * (180.0 / PI), theta_y_B * (180.0 / PI), theta_z_B * (180.0 / PI));
234         }
235         else return poseA; //Just return the same.
236     };
237 };
238
239 };
240 #endif /* CALIBRATION_H */

```

## 6.4 camera\_matrix.h

```

1 #ifndef CAMERA_MATRIX_H
2 #define CAMERA_MATRIX_H
3
4 #pragma once
5 #include "core/calibration.h"
6
7 class camera_matrix
8 {
9 public:
10     camera_matrix();
11     ~camera_matrix();
12     camera_matrix(Calibration calibration);
13
14 private:
15
16 };
17
18 #endif

```

## 6.5 data\_structures\_6D.h

```

1  #ifndef DATA_STRUCTURES_6D_H
2  #define DATA_STRUCTURES_6D_H
3
4  /*Standard*/
5  #include <algorithm>
6  #include "gpu/render_engine.cuh"
7
8  /*Header for Data Storage Class of DIRECT algorithm (basically a linked list)*/
9
10 /*Enum Structure for Directions*/
11 enum Direction {
12     X_DIRECTION, Y_DIRECTION, Z_DIRECTION,
13     XA_DIRECTION, YA_DIRECTION, ZA_DIRECTION
14 };
15
16 /*Point6D to store Pose Information*/
17 struct Point6D
18 {
19     Point6D(double xval, double yval, double zval, double xaval, double yaval, double zaval);
20     Point6D();
21     Point6D(gpu_cost_function::Pose p);
22
23     double x; double y; double z; double xa; double ya; double za;
24
25     double GetDistanceFrom(Point6D otherPoint);
26
27     Direction GetLargestDirection();
28
29     double GetDirection(Direction direction);
30
31     void UpdateDirection(Direction direction, double updated_value);
32 };
33
34 /*Storage Class (Linked List of HyperMatrices/Columns) for DIRECT optimization algorithm*/
35 struct HyperBox6D //Stores HyperCube Info
36 {
37     HyperBox6D(double value, Point6D center, Point6D sides);
38     HyperBox6D();
39
40     double value_;
41     double size_;
42
43     Point6D sides_;
44     Point6D center_;
45
46     void SetSides(Point6D new_sides);
47     Point6D GetSides();
48
49     bool containsPoint(Point6D point);
50
51     Point6D GetCenter();
52     void SetCenter(Point6D new_center);
53
54     /*Divide a Side in Three*/
55     void TrisectSide(Direction trisect_side);
56
57     void PrintCenter();
58
59 };
60
61 #endif /*DATA_STRUCTURES_6D_H*/

```

## 6.6 direct\_data\_storage.h

```

1  #ifndef DIRECT_DATA_STORAGE_H
2  #define DIRECT_DATA_STORAGE_H
3
4  /*Header for Data Storage Class of DIRECT algorithm (different (faster) than one in JTA but doesnt support
   Lipschitz search)*/
5
6  /*6D Data Structures*/
7  #include "data_structures_6D.h"
8
9  //Standard
10 #include <string>
11 #include <vector>
12
13 class DirectDataStorage {
14 public:
15     /*Initialize Direct Storage with a Unit Size HyperBox @ (.5, .5, .5, .5, .5, .5) with initial_value*/
16     DirectDataStorage(double initial_value);

```

```

17     DirectDataStorage(); /*Use Default Value of -1 For initial value*/
18     //~DirectDataStorage();
19
20     /*Remove and Add HyperBoxes*/
21     void AddHyperBox(HyperBox6D *new_box);
22     void DeleteHyperBoxes(std::vector<int> col_ids); /*Deletes the Best Hyperbox at the List of Column
IDs and Deletes Empty Columns*/
23
24     /*Get Number of Columns*/
25     unsigned int GetNumberColumns();
26
27     /*Get smallest Fvalue (last one) in column*/
28     HyperBox6D GetMinimumHyperbox(int col_id);
29
30     /*Get Smallest Fvalue (Last one) in Column*/
31     double GetMinimumHyperboxValue(int col_id);
32
33     /*Get Hyperbox Size Stored Column*/
34     double GetSizeStoredInColumn(int col_id);
35
36     /*Delete Contents of storage_matrix_ safely (also called in destructor)*/
37     void DeleteAllStoredHyperboxes();
38
39     /*Print Columns, Min/Max/Avg Column Length*/
40     void PrintSize();
41     /*Print */
42     void PrintContents();
43 private:
44     /*Vector of Vector of HyperBoxes:
45 Low Level Vector of HyperBoxes Represents All Hyperboxes of a Given Size, Kept in Sorted Decreasing (Max
Value @ 0) Order
46 High Level Vector of Vectors Represents All Sizes of Current Vectors, Kept in Sorted Increasing (Min
Value @ 0) Order*/
47     std::vector<std::vector<HyperBox6D*>*> storage_matrix_;
48
49     /*Vectors for Storing Minimum Hyperbox Size/Function Value Respectively for Each Column.
50 This is done for speed as it is much faster to access. Might need to reserve space in constructor. */
51     std::vector<double> minimum_value_columns_;
52     std::vector<double> size_columns_;
53
54 };
55
56 #endif /* DIRECT_DATA_STORAGE_H */

```

## 6.7 drr\_interactor.h

```

1 #pragma once
2 /*VTK*/
3
4 #include <vtkObjectFactory.h>
5 #include <vtkInteractorStyleTrackballActor.h>
6 #include <vtkRendererCollection.h>
7 #include <vtkTextActor.h>
8 #include <vtkTextProperty.h>
9 #include <vtkActor2DCollection.h>
10 #include <vtkPicker.h>
11 #include <vtkPropPicker.h>
12 #include <vtkProp.h>
13 #include <qcursor.h>
14
15 //Calibration To Convert Pose
16 #include "core/calibration.h"
17
18 /*Drr Tool Header*/
19 #include "gui/drr_tool.h"
20
21 /*DRR Globals*/
22 bool middleDownDRR = false; // Is CM button down?
23 bool leftDownDRR = false; //Is LM button down?
24 bool rightDownDRR = false; //Is RM button down
25 int rightDownDRRY = 0; //Y Pixel when RM Clicked
26 double rightDownDRRModelZ = 0; //Model's Z Translation when RM Clicked
27
28 class DRRInteractorStyle : public vtkInteractorStyleTrackballActor
29 {
30 public:
31     static DRRInteractorStyle* New();
32     vtkTypeMacro(DRRInteractorStyle, vtkInteractorStyleTrackballActor);
33
34     /*Pointer to Main Window*/
35     DRRTool* drrtool_;
36     void initialize_DRRTool(DRRTool* drrtool) {
37         drrtool_ = drrtool;

```

```

38     }
39
40     //Picked Function
41     bool ActivePick()
42     {
43         if (this->InteractionProp == NULL) return false;
44         else return true;
45     }
46
47     //KeyPress Turns Off Other Char Hotkeys
48     virtual void OnChar() {
49     }
50
51     //Keypress Function
52     virtual void OnKeyPress()
53     {
54         // Get the keypress
55         vtkRenderWindowInteractor *rwi = this->Interactor;
56         if (this->InteractionProp == NULL)
57         {
58             std::string key = rwi->GetKeySym();
59
60             this->Interactor->GetRenderWindow()->Render();
61             return;
62         }
63
64         vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
65         std::string key = rwi->GetKeySym();
66         double* Position = actor->GetPosition();
67
68         //Shift Class
69         if (rwi->GetShiftKey())
70         {
71             // Handle an arrow key
72             if (key == "Up")
73             {
74                 actor->RotateX(1);
75                 this->Interactor->GetRenderWindow()->Render();
76             }
77             // Handle an arrow key
78             if (key == "Down")
79             {
80                 actor->RotateX(-1 * 1);
81                 this->Interactor->GetRenderWindow()->Render();
82             }
83
84             // Handle an arrow key
85             if (key == "Left")
86             {
87                 actor->RotateY(-1 * 1);
88                 this->Interactor->GetRenderWindow()->Render();
89             }
90
91             // Handle an arrow key
92             if (key == "Right")
93             {
94                 actor->RotateY(1);
95                 this->Interactor->GetRenderWindow()->Render();
96             }
97         }
98         //Control Class
99         else if (rwi->GetControlKey())
100         {
101             // Handle an arrow key
102             if (key == "Up")
103             {
104                 actor->SetPosition(Position[0], Position[1], Position[2] + 1);
105                 this->Interactor->GetRenderWindow()->Render();
106             }
107             // Handle an arrow key
108             if (key == "Down")
109             {
110                 actor->SetPosition(Position[0], Position[1], Position[2] - 1);
111                 this->Interactor->GetRenderWindow()->Render();
112             }
113
114             // Handle an arrow key
115             if (key == "Left")
116             {
117                 actor->RotateZ(-1 * 1);
118                 this->Interactor->GetRenderWindow()->Render();
119             }
120
121             // Handle an arrow key
122             if (key == "Right")
123             {
124                 actor->RotateZ(1);

```

```

125         this->Interactor->GetRenderWindow()->Render();
126     }
127 }
128 //Naked Class
129 else
130 {
131     // Handle an arrow key
132     if (key == "Up")
133     {
134         actor->SetPosition(Position[0], Position[1] + 1, Position[2]);
135         this->Interactor->GetRenderWindow()->Render();
136     }
137     // Handle an arrow key
138     if (key == "Down")
139     {
140         actor->SetPosition(Position[0], Position[1] - 1, Position[2]);
141         this->Interactor->GetRenderWindow()->Render();
142     }
143     // Handle an arrow key
144     if (key == "Left")
145     {
146         actor->SetPosition(Position[0] - 1, Position[1], Position[2]);
147         this->Interactor->GetRenderWindow()->Render();
148     }
149     // Handle an arrow key
150     if (key == "Right")
151     {
152         actor->SetPosition(Position[0] + 1, Position[1], Position[2]);
153         this->Interactor->GetRenderWindow()->Render();
154     }
155 }
156
157 this->Interactor->GetRenderWindow()->Render();
158
159 //Forward events
160 vtkInteractorStyleTrackballActor::OnKeyPress();
161 }
162
163 //Left Mouse Down Function
164 virtual void OnLeftButtonDown()
165 {
166     leftDownDRR = true;
167
168     // Forward Events
169     vtkInteractorStyleTrackballActor::OnLeftButtonDown();
170 }
171
172 //Right Mouse Down Function
173 virtual void OnRightButtonDown()
174 {
175     rightDownDRR = true;
176     rightDownDRRY = QCursor::pos().y();
177
178     // Forward Events
179     vtkInteractorStyleTrackballActor::OnRightButtonDown();
180
181     if (this->InteractionProp == NULL)
182     {
183         return;
184     }
185     vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
186     rightDownDRRModelZ = actor->GetPosition()[2];
187 }
188
189 //Middle Mouse Down Funtion
190 virtual void OnMiddleButtonDown() {
191     middleDownDRR = true;
192
193     // Forward Events
194     vtkInteractorStyleTrackballActor::OnMiddleButtonDown();
195 }
196
197 //Left Mouse Up Function
198 virtual void OnLeftButtonUp()
199 {
200     if (this->InteractionProp == NULL)
201     {
202         return;
203     }
204     vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
205
206     leftDownDRR = false;
207     this->Interactor->GetRenderWindow()->Render();
208     // Forward Events
209     vtkInteractorStyleTrackballActor::OnLeftButtonUp();
210 }
211

```

```

212 //Right Mouse Up Function
213 virtual void OnRightButtonUp()
214 {
215     rightDownDRR = false;
216
217     // Forward Events
218     vtkInteractorStyleTrackballActor::OnRightButtonUp();
219 }
220
221 //Middle Mouse Up Function
222 virtual void OnMiddleButtonUp() {
223     middleDownDRR = false;
224
225     //Forward Events
226     vtkInteractorStyleTrackballActor::OnMiddleButtonUp();
227 }
228
229 //Mouse Movement
230 virtual void OnMouseMove()
231 {
232     if (this->InteractionProp == NULL) return;
233     if (leftDownDRR == true || rightDownDRR == true || middleDownDRR == true)
234     {
235         vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
236
237         //If Right Down and Not Left or MiddleScale The Z
238         if (!leftDownDRR && !middleDownDRR)
239         {
240             double* Position = actor->GetPosition();
241             actor->SetPosition(Position[0], Position[1], QCursor::pos().y() - rightDownDRRY +
rightDownDRRModelZ);
242         }
243         this->Interactor->GetRenderWindow()->Render();
244     }
245
246     // Forward Events
247     if (!rightDownDRR)
248         vtkInteractorStyleTrackballActor::OnMouseMove();
249
250     /*Draw DRR*/
251     drrtool_>DrawDRR();
252 }
253 };
254 vtkStandardNewMacro(DRRInteractorStyle);

```

## 6.8 frame.h

```

1 /*Header for Frame Class:
2 The Frame class stores:
3 1. Width and Height information for an x-ray image
4 2. Original x-ray image
5 3. Inverted grayscale of the x-ray image
6 3. Edge Detection constants (aperture, low/high threshold)
7 4. Edge Detected version of the image
8 5. maximum dilation constant
9 6. dilated version of the edge image*/
10
11 #ifndef FRAME_H
12 #define FRAME_H
13
14 /*Standard*/
15 #include <string>
16
17 /*OpenCV 3.1 Library*/
18 #include <opencv2/highgui/highgui.hpp>
19 #include <opencv2/imgproc/imgproc.hpp>
20
21 class Frame {
22 public:
23     Frame(std::string file_location, int aperture, int low_threshold,
24           int high_threshold, int dilation);
25     ~Frame(){};
26
27     /*Get Set Methods for Main Variables*/
28     /*Return Original Image*/
29     cv::Mat GetOriginalImage();
30     /*Return Edge Detected Image*/
31     cv::Mat GetEdgeImage();
32     /*Return Dilated Edge Detected Image*/
33     cv::Mat GetDilationImage();
34     /*Return Inverted Intensity Image*/
35     cv::Mat GetInvertedImage();
36

```

```

37     /*Store Public String Location*/
38     std::string file_location_;
39
40     /*Reset From Original (Resets Inverted/Segmented, Edge, Dilation from Original,
41 Useful if Trying to Reset from Segmentation)*/
42     void ResetFromOriginal();
43
44     /*Recalculate Edge Detected Image*/
45     void SetEdgeImage(int aperture, int low_threshold,
46         int high_threshold, bool use_reverse = false);
47     /*Recalculate Dilated Image*/
48     void SetDilatedImage(int dilation);
49
50     /*Get Canny Parameters*/
51     int GetAperture();
52     int GetHighThreshold();
53     int GetLowThreshold();
54 private:
55     /*Original Matrix*/
56     cv::Mat original_image_;
57     /*Edge Detected Matrix*/
58     cv::Mat edge_image_;
59     /*Dilation Matrix*/
60     cv::Mat dilation_image_;
61     /*Inverted Matrix (Store in Inverted)*/
62     cv::Mat inverted_image_;
63
64     /*Constants*/
65     int aperture_;
66     int low_threshold_;
67     int high_threshold_;
68     int dilation_;
69     int width_;
70     int height_;
71
72 };
73
74 #endif /* FRAME_H */

```

## 6.9 location\_storage.h

```

1  /*Location Storage Class is a Matrix for Different Model Poses for Each of the Different Loaded Images (Or
   Image Pairs if Optimized)*/
2  #ifndef LOCATION_STORAGE_H
3  #define LOCATION_STORAGE_H
4
5  /*Standard*/
6  #include <vector>
7
8  /*Direct Library*/
9  #include "data_structures_6D.h"
10
11 class LocationStorage {
12 public:
13     LocationStorage();
14     ~LocationStorage();
15
16     /*Add New Model to JTA-GPU So Initialize All Loaded Frames with
17 Default Pose (0,0,-.25*principal_distance / pixel_pitch,0,0,0)*/
18     void LoadNewModel(double principal_distance, double pixel_pitch);
19     /*Add New Frame to JTA-GPU So Initialize All Loaded Models with
20 Default Pose (0,0,-.25*principal_distance / pixel_pitch,0,0,0) for that Frame*/
21     void LoadNewFrame();
22
23     /*Access A Pose from Matrix*/
24     Point6D GetPose(int frame_index, int model_index);
25
26     /*Store a Pose to Matrix*/
27     void SavePose(int frame_index, int model_index, Point6D model_pose);
28
29     /*Get Frame Storage Size*/
30     int GetFrameCount();
31
32     /*Get Model Storage Size (-1 if Inconsistent Sizes)*/
33     int GetModelCount();
34 private:
35     /*List of Model Location Points for Each Frame
36 (So The Outside Vector is the length of the loaded frames
37 and the inside vector is the length of the loaded models)*/
38     std::vector<std::vector<Point6D>> location_storage_matrix_;
39
40     /*No Image Model Location Points
41 (This stores location points for models when there are no images

```

```

42 loaded, this is the size of the models loaded)*/
43     std::vector<Point6D> no_image_location_storage_vector;
44 };
45
46 #endif /* LOCATION_STORAGE_H */

```

## 6.10 machine\_learning\_tools.h

```

1 #pragma once
2
3 #include <torch/torch.h>
4 #include <torch/script.h>
5 #include <opencv2/highgui.hpp>
6 #include <opencv2/imgproc.hpp>
7 #include <cuda.h>
8 #include <cuda_runtime.h>
9 #include <device_launch_parameters.h>
10
11 cv::Mat segment_image(const cv::Mat& orig_image, bool black_sil_used, torch::jit::Module* model,
12                      unsigned int input_width, unsigned int input_height);
13
14

```

## 6.11 mainscreen\_size\_constants.h

```

1 /*Constants for resizing the spacings on the main screen*/
2
3 #ifndef MAINSCREEN_SIZE_CONSTANTS_H
4 #define MAINSCREEN_SIZE_CONSTANTS_H
5
6 /*Sizes for Main Window*/
7 int MINIMUM_WIDTH = 1600; // 1900;
8 int MINIMUM_HEIGHT = 918; // 900;
9
10 /*Minimum List Widget Size*/
11 const int MINIMUM_LIST_WIDGET_SIZE = 100;
12
13 /*Minimum Qvtk widget width*/
14 const int MINIMUM_QVTK_WIDGET_WIDTH = 831;
15
16 /*Border Paddings between object A and object B*/
17 const int GROUP_BOX_TO_BUTTON_PADDING_X = 25;
18 const int BUTTON_TO_BUTTON_PADDING_X = 11;
19 const int INSIDE_BUTTON_PADDING_X = 30;
20 const int INSIDE_RADIO_BUTTON_PADDING_X = 40;
21 const int APPLICATION_BORDER_TO_GROUP_BOX_PADDING_X = 55;
22 const int INSIDE_SPIN_BOX_PADDING_X = 25;
23 const int LABEL_TO_SPIN_BOX_PADDING_X = 15;
24 const int SPIN_BOX_TO_GROUP_BOX_PADDING_X = 60;
25 const int INSIDE_BUTTON_PADDING_RIGHT_COLUMN_X = 50;
26 const int GROUP_BOX_TO_QVTK_PADDING_X = 65;
27
28 const int GROUP_BOX_TO_GROUP_BOX_Y = 30;
29 const int GROUP_BOX_TO_BUTTON_PADDING_Y = 30;
30 const int BUTTON_TO_BUTTON_PADDING_Y = 11;
31 const int SPIN_BOX_TO_SPIN_BOX_PADDING_Y = 15;
32 const int INSIDE_BUTTON_PADDING_Y = 30;
33 const int INSIDE_RADIO_BUTTON_PADDING_Y = 10;
34 const int APPLICATION_BORDER_TO_GROUP_BOX_PADDING_Y = 40;
35 const int RADIO_BUTTON_TO_LIST_WIDGET_PADDING_Y = 25;
36 const int INSIDE_SPIN_BOX_PADDING_Y = 15;
37
38 /*Font Size*/
39 const int FONT_SIZE = 8;
40
41 #endif /* MAINSCREEN_SIZE_CONSTANTS_H */

```

## 6.12 metric\_enum.h

```

1 #ifndef METRIC_ENUM_H
2 #define METRIC_ENUM_H
3 /*Enumerator For Search Stage Flag*/
4 enum SearchStageFlag { Trunk = 0, Branch = 1, Leaf = 2 };
5
6 #endif /* METRIC_ENUM_H */

```



## 6.13 model.h

```

1  /*Header for Model Class Includes:
2  */
3
4  /*Standard*/
5  #include <string>
6  #include <vector>
7
8  /*VTK*/
9  #include <vtkSmartPointer.h>
10 #include <vtkSTLReader.h>
11
12 /*Custom STL Reader*/
13 #include "stl_reader.h"
14
15 #ifndef MODEL_H
16 #define MODEL_H
17
18 /*AS OF VERSION 3.3.1 SHOULD BE ABLE TO LOAD BOTH BINARY AND ASCII STL FILES*/
19 class Model
20 {
21 public:
22     Model(std::string file_location, std::string model_name, std::string model_type);
23     Model(){};
24     std::string file_location_; //Store File Location for Model
25     vtkSmartPointer<vtkSTLReader> cad_reader_; // Stores CAD model
26     std::vector<float> triangle_vertices_; //Vector of Triangle Vertices
27     std::vector<float> triangle_normals_; //Vector of Triangle Normals
28     /*Model Name: taken from prefix of file name. If duplicates a (x) is added*/
29     std::string model_name_;
30     /*Model Type: could be femur or implant or bone or type of bone, anything really...*/
31     std::string model_type_;
32     /*Bool indicating initialized correctly*/
33     bool initialized_correctly_;
34 private:
35     stl_reader::STL_STATUS LoadVerticesAndNormals();
36
37
38 };
39
40 #endif /* MODEL_H */

```

## 6.14 optimizer\_manager.h

```

1  /*Manages Optimization in a Seperate QT Thread*/
2
3  #ifndef OPTIMIZER_MANAGER_H
4  #define OPTIMIZER_MANAGER_H
5
6  /*Custom CUDA Headers*/
7  #include <gpu_model.cuh>
8  #include <gpu_intensity_frame.cuh>
9  #include <gpu_edge_frame.cuh>
10 #include <gpu_dilated_frame.cuh>
11 #include <gpu_metrics.cuh>
12 #include "calibration.h"
13
14 /*QT Threading*/
15 #include <qobject.h>
16 #include <qthread.h>
17 #include <QModelIndex>
18
19 /*Frame and Model and Location Storage*/
20 #include "frame.h"
21 #include "model.h"
22 #include "location_storage.h"
23
24 /*Direct Library*/
25 #include "data_structures_6D.h"
26 #include "direct_data_storage.h"
27
28 /*Custom Calibration Struct (Used in CUDA GPU METRICS)*/
29 #include "calibration.h"
30
31 /*Optimizer Settings*/
32 #include "optimizer_settings.h"
33
34 /*Metric Types*/
35 #include "metric_enum.h"
36
37 /*Cost Function Library*/
38 #include "cost_functions/CostFunctionManager.h"

```

```

39
40 #include "core/sym_trap_functions.h"
41
42 using namespace gpu_cost_function;
43
44 class OptimizerManager : public QObject
45 {
46     Q_OBJECT
47
48 public:
49     explicit OptimizerManager(QObject* parent = 0);
50     /*Sets up Everything for Optimizer and Also Handles CUDA Initialization, Can Fail!*/
51     bool Initialize(
52         QThread& optimizer_thread,
53         Calibration calibration_file,
54         std::vector<Frame> camera_A_frame_list, std::vector<Frame> camera_B_frame_list, unsigned int
current_frame_index,
55         std::vector<Model> model_list, QModelIndexList selected_models, unsigned int primary_model_index,
56         LocationStorage pose_matrix,
57         OptimizerSettings opt_settings,
58         jta_cost_function::CostFunctionManager trunk_manager, jta_cost_function::CostFunctionManager
branch_manager, jta_cost_function::CostFunctionManager leaf_manager,
59         QString opt_directive,
60         QString& error_message,
61         int iter_count);
62     ~OptimizerManager();
63
64
65     /* get cost numbers for symmetry plotting */
66     double EvaluateCostFunctionAtPoint(Point6D point, int stage);
67     void CalculateSymTrap();
68
69 signals:
70     /*Update Blue Current Optimum*/
71     void UpdateOptimum(double, double, double, double, double, double, double, unsigned int);
72     /*Finished*/
73     void finished();
74     /*Finished Optimizing Frame, Send Optimum to MainScreen, The last bool indicates if should move to
next frame*/
75     void OptimizedFrame(double, double, double, double, double, double, double, bool, unsigned int, bool,
QString);
76     /*Uh oh There was an Error. The string is the message*/
77     void OptimizerError(QString);
78     /*Update Display with Speed, Cost Function Calls, Current Minimum*/
79     void UpdateDisplay(double, int, double, unsigned int);
80     /*Update Dilation Background*/
81     void UpdateDilationBackground();
82
83     void CostFuncAtPoint(double);
84     void onUpdateOrientationSymTrap(double, double, double, double, double, double, double);
85     void onProgressBarUpdate(int);
86     void get_iter_count();
87
88 public slots:
89     /*Optimizer Biplane Single Model*/
90     void Optimize();
91
92     /*Emergency Stop*/
93     void onStopOptimizer();
94
95 private:
96
97     /*Initial Variables and Objects*/
98     /*Calibration File*/
99     Calibration calibration_;
100
101
102     /*Optimizer Settings*/
103     OptimizerSettings optimizer_settings_;
104
105     /*SYM TRAP SETTINGS*/
106     bool sym_trap_call;
107     //sym_trap *sym_trap_obj;
108
109     /*Frames*/
110     std::vector<Frame> frames_A_;
111     /*Camera B Frames*/
112     std::vector<Frame> frames_B_;
113
114     /*Models: All Models, Selected Non-Primary Models, and Primary Model*/
115     std::vector<Model> all_models_;
116     std::vector<Model> selected_non_primary_models_;
117     Model primary_model_;
118     /*Indices of All Selected Models*/
119     QModelIndexList selected_model_list_;
120     /*Index of Primary Model*/
121     unsigned int primary_model_index_;

```

```

122
123     /*Cost Function Managers For Each Stage*/
124     jta_cost_function::CostFunctionManager trunk_manager_;
125     jta_cost_function::CostFunctionManager branch_manager_;
126     jta_cost_function::CostFunctionManager leaf_manager_;
127
128     /*Should we progress to next frame?*/
129     bool progress_next_frame_;
130     /*Should we initialize with previous frame's best guess?*/
131     bool init_prev_frame_;
132     /*Index For Starting Frame in Optimization*/
133     unsigned int start_frame_index_;
134     unsigned int end_frame_index_;
135     int iter_count;
136
137     std::vector<int> img_indices_;
138
139     QString optimization_directive_;
140
141     void create_image_indices(std::vector<int>& img_indices, int start, int end);
142
143     /*Error Check*/
144     cudaError_t cuda_status_;
145
146     /*Correctly Initialized*/
147     bool successful_initialization_;
148
149     /*Dilation Values Based on Parameter Names (Dilation or DILATION or dilation) that are ints*/
150     int trunk_dilation_val_;
151     int branch_dilation_val_;
152     int leaf_dilation_val_;
153
154     /*Black Silhouette Values Based on Parameter Names (Black_Silhouette or Dark_Silhouette or
BLACK_SILHOUETTE or DARK_SILHOUETTE or black_silhouette or dark_silhouette)*/
155     bool trunk_dark_silhouette_val_;
156     bool branch_dark_silhouette_val_;
157     bool leaf_dark_silhouette_val_;
158
159     /*GPU Metrics Class*/
160     GPUMetrics* gpu_metrics_;
161
162     /*CUDA Cost Function Objects (Vector of GPU Models and vector of GPU Frames - note Dilated and
Intensity must have own vector
163 for each stage because their values could change with the stage from a black silhouette bool or a
dilation int)*/
164     /*Camera A (Monoplane or Biplane)*/
165     std::vector<GPUIntensityFrame*> gpu_intensity_frames_trunk_A_;
166     std::vector<GPUIntensityFrame*> gpu_intensity_frames_branch_A_;
167     std::vector<GPUIntensityFrame*> gpu_intensity_frames_leaf_A_;
168     std::vector<GPUEdgeFrame*> gpu_edge_frames_A_;
169     std::vector<GPUDilatedFrame*> gpu_dilated_frames_trunk_A_;
170     std::vector<GPUDilatedFrame*> gpu_dilated_frames_branch_A_;
171     std::vector<GPUDilatedFrame*> gpu_dilated_frames_leaf_A_;
172     /*Camera B (Biplane only)*/
173     std::vector<GPUIntensityFrame*> gpu_intensity_frames_trunk_B_;
174     std::vector<GPUIntensityFrame*> gpu_intensity_frames_branch_B_;
175     std::vector<GPUIntensityFrame*> gpu_intensity_frames_leaf_B_;
176     std::vector<GPUEdgeFrame*> gpu_edge_frames_B_;
177     std::vector<GPUDilatedFrame*> gpu_dilated_frames_trunk_B_;
178     std::vector<GPUDilatedFrame*> gpu_dilated_frames_branch_B_;
179     std::vector<GPUDilatedFrame*> gpu_dilated_frames_leaf_B_;
180
181     /*Models*/
182     GPUModel* gpu_principal_model_;
183     std::vector<GPUModel*> gpu_non_principal_models_;
184
185     /*Set Search Range*/
186     void SetSearchRange(Point6D range);
187
188     /*Set Search Range*/
189     void SetStartingPoint(Point6D starting_point);
190
191     /*Actual Range of Search Direction for Each Variable*/
192     Point6D range_;
193
194     /*Starting Point For Search*/
195     Point6D starting_point_;
196
197     /*Valid Search Range*/
198     bool valid_range_;
199
200     /*Budget*/
201     unsigned int budget_;
202
203     /*Data Storage*/
204     DirectDataStorage data_;
205

```

```

206  /*Potentially Optimal Column Ids (Given by Convex Hull)*/
207  std::vector<int> potentially_optimal_col_ids_;
208
209  /*Potentially Optimal Hyperboxes (Taken from potentially optimal column ids)*/
210  std::vector<HyperBox6D> potentially_optimal_hyperboxes_;
211
212  /*Convex Hull Loop of DIRECT*/
213  void ConvexHull();
214
215  /*Trisect Potentially Optimal Hypers and Sample and Add
216  to the storage. Delete old ones.*/
217  void TrisectPotentiallyOptimal();
218
219  /*Evaluate Cost Function at Given Point*/
220  double EvaluateCostFunction(Point6D point);
221
222  /*Denormalize Range Point (converts Unit Point to correct values)*/
223  Point6D DenormalizeRange(Point6D unit_point);
224
225  /*Denormalize Point From Center (converts Unit Point to correct values)*/
226  Point6D DenormalizeFromCenter(Point6D unit_point);
227
228  /*Cost Function Calls*/
229  unsigned int cost_function_calls_;
230
231  /*Lowest Min Value*/
232  double current_optimum_value_;
233
234  /*Argument (Location) of Lowest Min Value*/
235  Point6D current_optimum_location_;
236
237  /*Error Ocurrred*/
238  bool error_occurred_;
239
240  /*Clock for Timing Speed*/
241  /*(Milliseconds)*/
242  clock_t start_clock_, update_screen_clock_;
243
244  /*Store Post Matrix on Cost Functions*/
245  PoseMatrix pose_storage_;
246
247  /*Flag For Being in Either Trunk, Branch, or Z*/
248  unsigned int search_stage_flag_;
249
250 };
251
252 #endif /* OPTIMIZER_MANAGER_H */

```

## 6.15 optimizer\_settings.h

```

1  #ifndef OPTIMIZER_SETTINGS_H
2  #define OPTIMIZER_SETTINGS_H
3  /*Data Structures Used by All*/
4  #include "data_structures_6D.h"
5
6  /*Declare as MetaType So Can Send*/
7  #include <QMetaType>
8
9  /*Structure that Stores the Optimizer Settings Except for the Cost Function Information which is stored in
   the 3 Cost Function Managers*/
10 struct OptimizerSettings {
11     /*Constructor Destructor*/
12     OptimizerSettings();
13     ~OptimizerSettings();
14
15     /*Variables*/
16     /*Trunk*/
17     Point6D trunk_range;
18     int trunk_budget;
19
20     /*Branch*/
21     Point6D branch_range;
22     int branch_budget;
23     int number_branches;
24
25     /*Leaf*/
26     Point6D leaf_range;
27     int leaf_budget;
28
29     /*Optimizer Settings Which Stages Are on (trunk always on)*/
30     bool enable_branch_;
31     bool enable_leaf_;
32 };

```

```

33
34 Q_DECLARE_METATYPE(OptimizerSettings);
35
36 #endif /*OPTIMIZER_SETTINGS_H*/

```

## 6.16 settings\_constants.h

```

1 /*Constants for optimizer default settings and version*/
2
3 #ifndef SETTINGS_CONSTANTS_H
4 #define SETTINGS_CONSTANTS_H
5
6 /*Data Structures Used by All*/
7 #include "data_structures_6D.h"
8
9 /*Metric Type Enumerator*/
10 #include "metric_enum.h"
11
12 /*Version Numbers*/
13 const int VER_FIRST_NUM = 3;
14 const int VER_MIDDLE_NUM = 4;
15 const int VER_LAST_NUM = 0;
16
17 /*Variables*/
18 /*Trunk*/
19 const Point6D TRUNK_RANGE = Point6D(35,35,35,35,35,35);
20 const int TRUNK_BUDGET = 20000;
21 const int TRUNK_DILATION = 6;
22
23 /*Branches*/
24 const Point6D BRANCH_RANGE = Point6D(15,15,25,25,25,25);
25 const int BRANCH_BUDGET = 5000;
26 const int NUMBER_BRANCHES = 2;
27 const int BRANCH_DILATION_DECREASE = 2;
28
29 /*Z- SeaRCH*/
30 const Point6D Z_SEARCH_RANGE = Point6D(3,3,15,3,3,3);
31 const int Z_SEARCH_BUDGET = 5000;
32 const int Z_SEARCH_DILATION = 1;
33
34 /*Display Current Optimum During Optimization*/
35 const bool DISPLAY_CURRENT_OPTIMUM = true;
36
37 /*Optimizer Settings Control Window Other Stuff*/
38 const bool ENABLE_BRANCH = true;
39 const bool ENABLE_Z = true;
40 const bool SCALE_TRUNK = false;
41 const double SCALE_TRUNK_VALUE = 0.5;
42
43 /*Edge Constants Save*/
44 const int APERTURE = 3;
45 const int LOW_THRESH = 40;
46 const int HIGH_THRESH = 120;
47
48 /*Intensity Image Uses Black Silhouette? (True = Black, False = White)*/
49 const bool BLACK_SILHOUETTE = true;
50
51 /*Edge and Intensity Weights If Combined*/
52 const double INTENSITY_WEIGHT = 1;
53 const double EDGE_WEIGHT = 1;
54
55 #endif /* SETTINGS_CONSTANTS_H */

```

## 6.17 settings\_window\_size\_constants.h

```

1 /*Constants for resizing the spacings on the main screen*/
2
3 #ifndef SETTINGS_WINDOW_SIZE_CONSTANTS_H
4 #define SETTINGS_WINDOW_SIZE_CONSTANTS_H
5
6
7 /*Border Paddings between object A and object B*/
8 const int BUTTON_TO_BUTTON_PADDING_X = 35;
9 const int INSIDE_BUTTON_PADDING_X = 60;
10 const int INSIDE_RADIO_BUTTON_PADDING_X = 35;
11 const int APPLICATION_BORDER_TO_GROUP_BOX_PADDING_X = 55;
12 const int INSIDE_SPIN_BOX_PADDING_X = 25;
13 const int LABEL_TO_SPIN_BOX_PADDING_X = 15;
14 const int SPIN_BOX_TO_LABEL_PADDING_X = 25;

```

```

15 const int SMALL_GROUP_BOX_PADDING_X = 30;
16 const int GROUP_BOX_TO_SMALL_GROUP_BOX_X = 30;
17 const int GROUP_BOX_TO_GROUP_BOX_X = 30;
18 const int GROUP_BOX_TO_RADIO_BUTTON_X = 40;
19 const int BIG_GROUP_BOX_TO_SPIN_BOX_X = 115;
20
21 const int SMALL_GROUP_BOX_PADDING_Y = 30;
22 const int GROUP_BOX_TO_SMALL_GROUP_BOX_Y = 30;
23 const int CHECKBOX_TO_LABEL_Y = 30;
24 const int LABEL_TO_LABEL_PADDING_Y = 25;
25 const int GROUP_BOX_TO_GROUP_BOX_Y = 35;
26 const int SMALL_GROUP_BOX_TO_GROUP_BOX_Y = 30;
27 const int GROUP_BOX_TO_LABEL_PADDING_Y = 30;
28 const int GROUP_BOX_TO_RADIO_BUTTON_PADDING_Y = 30;
29 const int SPIN_BOX_TO_SPIN_BOX_PADDING_Y = 15;
30 const int INSIDE_BUTTON_PADDING_Y = 30;
31 const int INSIDE_RADIO_BUTTON_PADDING_Y = 10;
32 const int APPLICATION_BORDER_TO_GROUP_BOX_PADDING_Y = 30;
33 const int INSIDE_SPIN_BOX_PADDING_Y = 15;
34
35
36 #endif /* SETTINGS_WINDOW_SIZE_CONSTANTS_H */

```

## 6.18 stl\_reader.h

```

1 #pragma once
2 /*QT Headers*/
3 #include <QtCore\qstring.h>
4 #include <QtCore\qfile.h>
5 #include <QtCore\qfileinfo.h>
6 #include <QtCore\qtextstream.h>
7
8 /*Standard Library*/
9 #include <vector>
10
11 namespace stl_reader {
12
13     /*ENUM for STL file status*/
14     enum STL_STATUS { STL_INVALID, STL_ASCII, STL_BINARY };
15
16     /*Function to determine if file is a valid STL file and, if so, whether it is binary or ascii*/
17     STL_STATUS getStlFileFormat(const QString &path);
18
19     /*STL reader function (binary or ascii)
20     Populates two vector<floats>, one contains the triangle vertices, the other contains the triangle normals
21     */
22     STL_STATUS readAnySTL(const QString &path, std::vector<float> &triangleVertices, std::vector<float> &triangleNormals);
23 }

```

## 6.19 STLReader.h

```

1 #pragma once
2 /*QT Headers*/
3 #include <QtCore\qstring.h>
4 #include <QtCore\qfile.h>
5 #include <QtCore\qfileinfo.h>
6 #include <QtCore\qtextstream.h>
7
8 /*Standard Library*/
9 #include <vector>
10
11 namespace stl_reader_BIG {
12
13     /*ENUM for STL file status*/
14     enum STL_STATUS { STL_INVALID, STL_ASCII, STL_BINARY };
15
16     /*Function to determine if file is a valid STL file and, if so, whether it is binary or ascii*/
17     STL_STATUS getStlFileFormat(const QString &path);
18
19     /*STL reader function (binary or ascii)
20     Populates a vector of two vector<floats>, one contains the triangle vertices, the other contains the
21     triangle normals
22     */
23     STL_STATUS readAnySTL(const QString &path, std::vector<std::vector<float>> &stl_storage);
24 }

```

## 6.20 sym\_trap\_functions.h

```

1 #pragma once
2
3 #include "core/data_structures_6D.h"
4 #include <cmath>
5 #include <vector>
6 #include <sstream>
7
8 Point6D compute_mirror_pose(Point6D point);
9 void matmult4(float ans[4][4], float matrix1[4][4], float matrix2[4][4]);
10 void matmult3(float ans[3][3], const float matrix1[3][3], const float matrix2[3][3]);
11 void invert_transform(float result[4][4], const float tran[4][4]);
12 void equivalent_axis_angle_rotation(float rot[3][3], const float m[3], const float angle);
13 void cross_product(float CP[3], const float v1[3], const float v2[3]);
14 void dot_product(float& result, const float vector1[3], const float vector2[3]);
15 void rotation_matrix(float R[3][3], Point6D pose);
16 void create_3l2_transform(float transform[4][4], Point6D pose);
17 void getRotations3l2(float& xr, float& yr, float& zr, const float Rot[3][3]);
18
19 void copy_matrix_by_value(float(&new_matrix)[3][3], const float(&old_matrix)[3][3]);
20 void create_vector_of_poses(std::vector<Point6D>& pose_list, Point6D pose, int numPoses);
21
22 template<typename T>
23 std::vector<double> static linspace(T start_in, T end_in, int num_in);
24
25

```

## 6.21 CostFunction.h

```

1 #pragma once
2
3 /*Cost Function Parameters*/
4 #include "Parameter.h"
5
6 /*Standard Library*/
7 #include <vector>
8 #include <string>
9
10 namespace jta_cost_function {
11
12     class CostFunction {
13     public:
14         /*Constructor*/
15         __declspec(dllexport) CostFunction();
16         __declspec(dllexport) CostFunction(std::string cost_function_name);
17         __declspec(dllexport) ~CostFunction();
18
19         /*Add Parameter (w/ Default Value)*/
20         __declspec(dllexport) void addParameter(Parameter<double> new_parameter);
21         __declspec(dllexport) void addParameter(Parameter<int> new_parameter);
22         __declspec(dllexport) void addParameter(Parameter<bool> new_parameter);
23
24         /*Set Parameter Values (Bool for Success)*/
25         __declspec(dllexport) bool setDoubleParameterValue(std::string parameter_name, double value);
26         __declspec(dllexport) bool setIntParameterValue(std::string parameter_name, int value);
27         __declspec(dllexport) bool setBoolParameterValue(std::string parameter_name, bool value);
28
29         /*Get Parameter Values (Bool for Success)*/
30         __declspec(dllexport) bool getDoubleParameterValue(std::string parameter_name, double &value);
31         __declspec(dllexport) bool getIntParameterValue(std::string parameter_name, int &value);
32         __declspec(dllexport) bool getBoolParameterValue(std::string parameter_name, bool &value);
33
34         /*Get Parameters by Type Groups*/
35         __declspec(dllexport) std::vector<Parameter<double>> getDoubleParameters();
36         __declspec(dllexport) std::vector<Parameter<int>> getIntParameters();
37         __declspec(dllexport) std::vector<Parameter<bool>> getBoolParameters();
38
39         /*Get/Set Cost Function Name*/
40         __declspec(dllexport) std::string getCostFunctionName();
41         __declspec(dllexport) void setCostFunctionName(std::string cost_function_name);
42
43     private:
44         /*Containers for Parameters*/
45         std::vector<Parameter<double>> double_parameters_;
46         std::vector<Parameter<int>> int_parameters_;
47         std::vector<Parameter<bool>> bool_parameters_;
48
49         /*Cost Function Name*/
50         std::string cost_function_name_;
51     };
52 }
53
54

```

## 6.22 CostFunctionManager.h

```

1  #ifndef COSTFUNCTIONMANAGER_H
2  #define COSTFUNCTIONMANAGER_H
3
4  /*Class for Storing Cost Function Info*/
5  #include "CostFunction.h"
6
7  /*Cost Function Tools Library*/
8  #include "gpu/gpu_image.cuh"
9  #include "gpu/gpu_frame.cuh"
10 #include "gpu/gpu_dilated_frame.cuh"
11 #include "gpu/gpu_edge_frame.cuh"
12 #include "gpu/gpu_intensity_frame.cuh"
13 #include "gpu/gpu_model.cuh"
14 #include "gpu/gpu_metrics.cuh"
15 #include "gpu/render_engine.cuh"
16
17 /*Stage Enum*/
18 #include "Stage.h"
19
20 /*Standard Library*/
21 #include <vector>
22
23 namespace jta_cost_function {
24     class CostFunctionManager {
25     public:
26         /*****
27         /*****PUBLIC DLL FUNCTIONS BEGIN *****/
28         /***** (DO NOT EDIT) *****/
29         /*****
30         /*Constructor
31         Called once when the client initially loads and populates the list of available
32         cost functions. There will be three instances, one for each stage of DIRECT.
33         Also sets an active cost function (default is the DIRECT_DILATION).
34         The parameters are all default. To load previously saved session parameters, the constructor
35         for the client will call the updateCostFunctionParameterValues(...)*/
36         __declspec(dllexport) CostFunctionManager(Stage stage);
37         __declspec(dllexport) CostFunctionManager();
38         __declspec(dllexport) ~CostFunctionManager();
39
40         /*Set Active Cost Function*/
41         __declspec(dllexport) void setActiveCostFunction(std::string cost_function_name);
42
43         /*Update Cost Function Values from Saved Session*/
44         __declspec(dllexport) bool updateCostFunctionParameterValues(std::string cost_function_name,
45         std::string parameter_name, double value);
46         __declspec(dllexport) bool updateCostFunctionParameterValues(std::string cost_function_name,
47         std::string parameter_name, int value);
48         __declspec(dllexport) bool updateCostFunctionParameterValues(std::string cost_function_name,
49         std::string parameter_name, bool value);
50
51         /*Call Initialization for Active Cost Function*/
52         __declspec(dllexport) bool InitializeActiveCostFunction(std::string &error_message);
53
54         /*Call Destructor for Active Cost Function*/
55         __declspec(dllexport) bool DestructActiveCostFunction(std::string &error_message);
56
57         /*Call Active Cost Function*/
58         __declspec(dllexport) double callActiveCostFunction();
59
60         /*Get Active Cost Function*/
61         __declspec(dllexport) std::string getActiveCostFunction();
62
63         /*Get Active Cost Function Class*/
64         __declspec(dllexport) CostFunction* getActiveCostFunctionClass();
65
66         /*Get Cost Function Class*/
67         __declspec(dllexport) CostFunction* getCostFunctionClass(std::string cost_function_name);
68
69         /*Get Vector of Cost Functions*/
70         __declspec(dllexport) std::vector<CostFunction> getAvailableCostFunctions();
71
72         /*Set Current Frame Index*/
73         __declspec(dllexport) void setCurrentFrameIndex(unsigned int current_frame_index);
74
75         /*Upload Data (Images, Poses etc.)*/
76         __declspec(dllexport) void UploadData(std::vector<gpu_cost_function::GPUEdgeFrame*>*
77         gpu_edge_frames_A,
78         std::vector<gpu_cost_function::GPUTilatedFrame*>* gpu_dilated_frames_A,
79         std::vector<gpu_cost_function::GPUIntensityFrame*>* gpu_intensity_frames_A,
80         std::vector<gpu_cost_function::GPUEdgeFrame*>* gpu_edge_frames_B,
81         std::vector<gpu_cost_function::GPUTilatedFrame*>* gpu_dilated_frames_B,
82         std::vector<gpu_cost_function::GPUIntensityFrame*>* gpu_intensity_frames_B,
83         gpu_cost_function::GPUModel* gpu_principal_model,
84         std::vector<gpu_cost_function::GPUModel*>* gpu_non_principal_models,

```



```

82     gpu_cost_function::GPUMetrics* gpu_metrics,
83     PoseMatrix* pose_storage,
84     bool biplane_mode);
85 /*****
86 /*****PUBLIC DLL FUNCTIONS END *****/
87 /*****
88
89     private:
90 /*****
91     /* *****/
92 /***** (DO NOT EDIT) *****/
93 /*****
94
95     /*List Cost Functions
96 In this function a cost function that will be loaded to the client and optimizer
97 must be listed by name. The parameters should also be specified.*/
98     void listCostFunctions();
99
100     /*Vector of Cost Functions*/
101     std::vector<CostFunction> available_cost_functions_;
102
103     /*Active Cost Function*/
104     std::string active_cost_function_;
105
106 /*****
107 /*****ESSENTIAL CLASS VARIABLES END *****/
108 /*****
109
110
111 /*****
112 /*****COST FUNCTION VARIABLES BEGIN *****/
113 /***** (DO NOT EDIT) *****/
114 /*****
115
116     /*Stage Enum*/
117     Stage stage_;
118
119     /*Storage for GPU Metrics class*/
120     gpu_cost_function::GPUMetrics* gpu_metrics_;
121
122     /*Storage for Data (images, poses ,etc.)*/
123     /*Pointer to Vector of GPU Frame Pointers*/
124     /*Camera A*/
125     std::vector<gpu_cost_function::GPUEdgeFrame*> gpu_edge_frames_A_;
126     std::vector<gpu_cost_function::GPUDilatedFrame*> gpu_dilated_frames_A_;
127     std::vector<gpu_cost_function::GPUIntensityFrame*> gpu_intensity_frames_A_;
128     /*Camera B*/
129     std::vector<gpu_cost_function::GPUEdgeFrame*> gpu_edge_frames_B_;
130     std::vector<gpu_cost_function::GPUDilatedFrame*> gpu_dilated_frames_B_;
131     std::vector<gpu_cost_function::GPUIntensityFrame*> gpu_intensity_frames_B_;
132
133     /*Pointer to Vector of principal GPU Model Pointer*/
134     gpu_cost_function::GPUModel* gpu_principal_model_;
135     /*Pointer to Vector of non-principal GPU Model Pointers*/
136     std::vector<gpu_cost_function::GPUModel*> gpu_non_principal_models_;
137     float* prin_dist_;
138     /*Current Frame Index (0 based)*/
139     unsigned int current_frame_index_;
140
141     /*Pose Matrix*/
142     PoseMatrix* pose_storage_;
143
144     /*Biplane Mode?*/
145     bool biplane_mode_;
146
147 /*****
148 /*****COST FUNCTION VARIABLES END*****/
149 /*****
150
151
152
153 /*****
154 /*****CUSTOM COST FUNCTION VARIABLES BEGIN *****/
155 /***** (DO NOT EDIT) *****/
156 /*****
157     /*HEADERS THAT INTERACT WITH WIZARD*/
158     /*Custom Variable Headers for Cost Functions*/
159 #include "sym_trap_functionCustomVariables.h"
160 #include "DD_NEW_POLE_CONSTRAINTCustomVariables.h"
161 #include "DIRECT_DILATION_POLE_CONSTRAINTCustomVariables.h"
162 #include "DIRECT_DILATION_SAME_ZCustomVariables.h"
163 #include "DIRECT_DILATION_TlCustomVariables.h"
164 #include "DIRECT_DILATIONCustomVariables.h"
165 #include "DIRECT_MAHFOUZCustomVariables.h"
166     /*END HEADERS THAT INTERACT WITH WIZARD*/
167 /*****
168 /*****CUSTOM COST FUNCTION VARIABLES END*****/

```

```

169 /*****
170
171
172
173
174 /***** WARNING *****/
175 /*****
176 /*****DO NOT EDIT FUNCTIONS BELOW *****/
177 /*****
178     /*FUNCTIONS THAT INTERACT WITH WIZARD*/
179     /*Cost Function Implementations*/
180     double costFunctionsym_trap_function();
181     double costFunctionDD_NEW_POLE_CONSTRAINT();
182     double costFunctionDIRECT_DILATION_POLE_CONSTRAINT();
183     double costFunctionDIRECT_DILATION_SAME_Z();
184     double costFunctionDIRECT_DILATION_Tl();
185     double costFunctionDIRECT_DILATION();
186     double costFunctionDIRECT_MAHFOUZ();
187     /*Cost Function Initializations*/
188     bool initializesym_trap_function(std::string& error_message);
189     bool initializeDD_NEW_POLE_CONSTRAINT(std::string& error_message);
190     bool initializeDIRECT_DILATION_POLE_CONSTRAINT(std::string& error_message);
191     bool initializeDIRECT_DILATION_SAME_Z(std::string& error_message);
192     bool initializeDIRECT_DILATION_Tl(std::string& error_message);
193     bool initializeDIRECT_DILATION(std::string& error_message);
194     bool initializeDIRECT_MAHFOUZ(std::string& error_message);
195     /*Cost Function Destructors*/
196     bool destructsym_trap_function(std::string& error_message);
197     bool destructDD_NEW_POLE_CONSTRAINT(std::string& error_message);
198     bool destructDIRECT_DILATION_POLE_CONSTRAINT(std::string& error_message);
199     bool destructDIRECT_DILATION_SAME_Z(std::string& error_message);
200     bool destructDIRECT_DILATION_Tl(std::string& error_message);
201     bool destructDIRECT_DILATION(std::string& error_message);
202     bool destructDIRECT_MAHFOUZ(std::string& error_message);
203     /*END FUNCTIONS THAT INTERACT WITH WIZARD*/
204 /***** END WARNING *****/
205 /*****
206 /*****DO NOT EDIT FUNCTIONS ABOVE *****/
207 /*****
208     };
209 }
210
211 #endif //COSTFUNCTIONMANAGER_H

```

## 6.23 DD\_NEW\_POLE\_CONSTRAINTCustomVariables.h

```

1 #pragma once
2 /*****Headers*****/
3 /*Cost Function Tools Library*/
4 #include "gpu/gpu_image.cuh"
5 #include "gpu/gpu_frame.cuh"
6 #include "gpu/gpu_dilated_frame.cuh"
7 #include "gpu/gpu_edge_frame.cuh"
8 #include "gpu/gpu_intensity_frame.cuh"
9 #include "gpu/gpu_model.cuh"
10 #include "gpu/gpu_metrics.cuh"
11 #include "gpu/render_engine.cuh"
12 /*Stage Enum*/
13 #include "Stage.h"
14 /*Parameter Class*/
15 #include "Parameter.h"
16
17 /*****Begin Custom Variables*****/
18 double x_loc_non;
19 double z_loc_non;

```

## 6.24 DIRECT\_DILATION\_POLE\_CONSTRAINTCustomVariables.h

```

1 #pragma once
2 /*****Headers*****/
3 /*Cost Function Tools Library*/
4 #include "gpu/gpu_image.cuh"
5 #include "gpu/gpu_frame.cuh"
6 #include "gpu/gpu_dilated_frame.cuh"
7 #include "gpu/gpu_edge_frame.cuh"
8 #include "gpu/gpu_intensity_frame.cuh"
9 #include "gpu/gpu_model.cuh"
10 #include "gpu/gpu_metrics.cuh"
11 #include "gpu/render_engine.cuh"

```

```

12 /*Stage Enum*/
13 #include "Stage.h"
14 /*Parameter Class*/
15 #include "Parameter.h"
16
17 /*****Begin Custom Variables*****/
18 /*Sum of the white pixels in the current dilation comparison image*/
19
20

```

## 6.25 DIRECT\_DILATION\_SAME\_ZCustomVariables.h

```

1 #pragma once
2 /*****Headers*****/
3 /*Cost Function Tools Library*/
4 #include "gpu/gpu_image.cuh"
5 #include "gpu/gpu_frame.cuh"
6 #include "gpu/gpu_dilated_frame.cuh"
7 #include "gpu/gpu_edge_frame.cuh"
8 #include "gpu/gpu_intensity_frame.cuh"
9 #include "gpu/gpu_model.cuh"
10 #include "gpu/gpu_metrics.cuh"
11 #include "gpu/render_engine.cuh"
12 /*Stage Enum*/
13 #include "Stage.h"
14 /*Parameter Class*/
15 #include "Parameter.h"
16
17 /*****Begin Custom Variables*****/
18 /*Sum of the white pixels in the current dilation comparison image*/
19 int DIRECT_DILATION_SAME_Z_current_white_pix_sum_dilated_comparison_image_A;
20 int DIRECT_DILATION_SAME_Z_current_white_pix_sum_dilated_comparison_image_B;
21 int DIRECT_DILATION_SAME_Z_current_dilation_parameter;
22 double DIRECT_DILATION_SAME_Z_current_z_weight_parameter;

```

## 6.26 DIRECT\_DILATION\_T1CustomVariables.h

```

1 #pragma once
2 /*****Headers*****/
3 /*Cost Function Tools Library*/
4 #include "gpu/gpu_image.cuh"
5 #include "gpu/gpu_frame.cuh"
6 #include "gpu/gpu_dilated_frame.cuh"
7 #include "gpu/gpu_edge_frame.cuh"
8 #include "gpu/gpu_intensity_frame.cuh"
9 #include "gpu/gpu_model.cuh"
10 #include "gpu/gpu_metrics.cuh"
11 #include "gpu/render_engine.cuh"
12 /*Stage Enum*/
13 #include "Stage.h"
14 /*Parameter Class*/
15 #include "Parameter.h"
16
17 /*****Begin Custom Variables*****/
18 /*Sum of the white pixels in the current dilation comparison image*/
19 int DIRECT_DILATION_T1_current_white_pix_sum_dilated_comparison_image_A;
20 int DIRECT_DILATION_T1_current_white_pix_sum_dilated_comparison_image_B;
21 int DIRECT_DILATION_T1_current_dilation_parameter;

```

## 6.27 DIRECT\_DILATIONCustomVariables.h

```

1 #pragma once
2 /*****Headers*****/
3 /*Cost Function Tools Library*/
4 #include "gpu/gpu_image.cuh"
5 #include "gpu/gpu_frame.cuh"
6 #include "gpu/gpu_dilated_frame.cuh"
7 #include "gpu/gpu_edge_frame.cuh"
8 #include "gpu/gpu_intensity_frame.cuh"
9 #include "gpu/gpu_model.cuh"
10 #include "gpu/gpu_metrics.cuh"
11 #include "gpu/render_engine.cuh"
12 /*Stage Enum*/
13 #include "Stage.h"
14 /*Parameter Class*/

```

```

15 #include "Parameter.h"
16
17 /*****Begin Custom Variables*****/
18 /*Sum of the white pixels in the current dilation comparison image*/
19 int DIRECT_DILATION_current_white_pix_sum_dilated_comparison_image_A_;
20 int DIRECT_DILATION_current_white_pix_sum_dilated_comparison_image_B_;
21 int DIRECT_DILATION_current_dilation_parameter;

```

## 6.28 DIRECT\_MAHFOUZCustomVariables.h

```

1 #pragma once
2 /*****Headers*****/
3 /*Cost Function Tools Library*/
4 #include "gpu/gpu_image.cuh"
5 #include "gpu/gpu_frame.cuh"
6 #include "gpu/gpu_dilated_frame.cuh"
7 #include "gpu/gpu_edge_frame.cuh"
8 #include "gpu/gpu_intensity_frame.cuh"
9 #include "gpu/gpu_model.cuh"
10 #include "gpu/gpu_metrics.cuh"
11 #include "gpu/render_engine.cuh"
12 /*Stage Enum*/
13 #include "Stage.h"
14 /*Parameter Class*/
15 #include "Parameter.h"
16
17 /*****Begin Custom Variables*****/

```

## 6.29 Parameter.h

```

1 #ifndef PARAMETER_H
2 #define PARAMETER_H
3 /*Parameter Class Header*/
4 /* Info: The cost function class contains a vector of parameter classes which represent
5 any parameters (must be either double, integer, or bool) that the cost function might require.
6 These cost function parameters are added to the parameter storage vector in the constructor of
7 the cost function class. Values for the parameters can be set from the JTA client and are
8 saved between sessions. Default values for each parameter must be provided in the constructor
9 for the cost function along with a parameter type and parameter name.*/
10
11 /*Standard Library*/
12 #include <string>
13 #include <type_traits>
14
15 /*Custom Namespace for JTA Cost Function Library (Compiling as DLL)*/
16 namespace jta_cost_function {
17
18     template <typename Parameter_Type>
19     class Parameter {
20     public:
21         static_assert((std::is_same<double, Parameter_Type>::value
22             || std::is_same<int, Parameter_Type>::value
23             || std::is_same<bool, Parameter_Type>::value), "Parameter type must be double, int, or
24             bool!");
25     };
26
27     template <>
28     class Parameter<double> {
29     public:
30         /*Constructors*/
31         __declspec(dllexport) Parameter() {
32             parameter_name_ = "Nameless Parameter";
33             parameter_value_ = 0;
34             parameter_type_ = "DOUBLE";
35         };
36         __declspec(dllexport) Parameter(std::string parameter_name, double parameter_value) {
37             parameter_name_ = parameter_name;
38             parameter_value_ = parameter_value;
39             parameter_type_ = "DOUBLE";
40         };
41
42         /*Methods*/
43         /*Get Parameter Name*/
44         __declspec(dllexport) std::string getParameterName() {
45             return parameter_name_;
46         };
47
48         /*Get/Set Parameter Value*/
49         __declspec(dllexport) double getParameterValue() {
50             return parameter_value_;
51         };
52     };
53
54 }
55 #endif

```

```

49         };
50         __declspec(dllexport) void setParameterValue(double parameter_value) {
51             parameter_value_ = parameter_value;
52         };
53
54         /*Get Class Type*/
55         __declspec(dllexport) std::string getParameterType() {
56             return parameter_type_;
57         };
58
59     private:
60         /*Variables*/
61         /*Parameter Name*/
62         std::string parameter_name_;
63
64         /*Parameter Value*/
65         double parameter_value_;
66
67         /*Class Type*/
68         std::string parameter_type_;
69     };
70
71     template <>
72     class Parameter<int> {
73     public:
74         /*Constructors*/
75         __declspec(dllexport) Parameter() {
76             parameter_name_ = "Nameless Parameter";
77             parameter_value_ = 0;
78             parameter_type_ = "INT";
79         };
80         __declspec(dllexport) Parameter(std::string parameter_name, int parameter_value) {
81             parameter_name_ = parameter_name;
82             parameter_value_ = parameter_value;
83             parameter_type_ = "INT";
84         };
85
86         /*Methods*/
87         /*Get Parameter Name*/
88         __declspec(dllexport) std::string getParameterName() {
89             return parameter_name_;
90         };
91
92         /*Get/Set Parameter Value*/
93         __declspec(dllexport) int getParameterValue() {
94             return parameter_value_;
95         };
96         __declspec(dllexport) void setParameterValue(int parameter_value) {
97             parameter_value_ = parameter_value;
98         };
99
100        /*Get Class Type*/
101        __declspec(dllexport) std::string getParameterType() {
102            return parameter_type_;
103        };
104
105    private:
106        /*Variables*/
107        /*Parameter Name*/
108        std::string parameter_name_;
109
110        /*Parameter Value*/
111        int parameter_value_;
112
113        /*Class Type*/
114        std::string parameter_type_;
115    };
116
117     template <>
118     class Parameter<bool> {
119     public:
120         /*Constructors*/
121         __declspec(dllexport) Parameter() {
122             parameter_name_ = "Nameless Parameter";
123             parameter_value_ = 0;
124             parameter_type_ = "BOOL";
125         };
126         __declspec(dllexport) Parameter(std::string parameter_name, bool parameter_value) {
127             parameter_name_ = parameter_name;
128             parameter_value_ = parameter_value;
129             parameter_type_ = "BOOL";
130         };
131
132         /*Methods*/
133         /*Get Parameter Name*/
134         __declspec(dllexport) std::string getParameterName() {
135             return parameter_name_;

```

```

136         };
137
138         /*Get/Set Parameter Value*/
139         __declspec(dllexport) bool getParameterValue() {
140             return parameter_value_;
141         };
142         __declspec(dllexport) void setParameterValue(bool parameter_value) {
143             parameter_value_ = parameter_value;
144         };
145
146         /*Get Class Type*/
147         __declspec(dllexport) std::string getParameterType() {
148             return parameter_type_;
149         };
150
151     private:
152         /*Variables*/
153         /*Parameter Name*/
154         std::string parameter_name_;
155
156         /*Parameter Value*/
157         bool parameter_value_;
158
159         /*Class Type*/
160         std::string parameter_type_;
161     };
162 }
163
164 #endif //PARAMETER_H

```

## 6.30 Stage.h

```

1 #ifndef STAGE_H
2 #define STAGE_H
3
4 /*Enum Class for Stages*/
5 enum class Stage { Trunk, Branch, Leaf };
6
7 #endif /*STAGE_H*/

```

## 6.31 sym\_trap\_functionCustomVariables.h

```

1 #pragma once
2 /*****Headers*****/
3 /*Cost Function Tools Library*/
4 #include "gpu/gpu_image.cuh"
5 #include "gpu/gpu_frame.cuh"
6 #include "gpu/gpu_dilated_frame.cuh"
7 #include "gpu/gpu_edge_frame.cuh"
8 #include "gpu/gpu_intensity_frame.cuh"
9 #include "gpu/gpu_model.cuh"
10 #include "gpu/gpu_metrics.cuh"
11 #include "gpu/render_engine.cuh"
12 /*Stage Enum*/
13 #include "Stage.h"
14 /*Parameter Class*/
15 #include "Parameter.h"
16
17 /*****Begin Custom Variables*****/
18 void invert_transformation(float result[4][4], float tran[4][4])
19 {
20     int i, j;
21     /* Upper left 3x3 of result is transpose of upper left 3x3 of tran. */
22     for (i = 0; i < 3; ++i)
23         for (j = 0; j < 3; ++j)
24             result[i][j] = tran[j][i];
25     /* Set the values for the last column of the result */
26     result[3][0] = result[3][1] = result[3][2] = 0.0;
27     result[3][3] = 1.0;
28     /* Initialize the values of the last column of the result. */
29     result[0][3] = result[1][3] = result[2][3] = 0.0;
30     for (i = 0; i < 3; i++) {
31         for (j = 0; j < 3; j++) {
32             result[i][3] -= result[i][j] * tran[j][3];
33         }
34     }
35 }
36
37 void matmult(float ans[4][4], float matrix1[4][4], float matrix2[4][4])

```

```

38 {
39     int i, j, k;
40     for (i = 0; i < 4; i++)
41         for (j = 0; j < 4; j++)
42             ans[i][j] = 0.0;
43     for (i = 0; i < 4; i++)
44         for (j = 0; j < 4; j++)
45             for (k = 0; k < 4; k++)
46                 ans[i][j] += matrix1[i][k] * matrix2[k][j];
47 }
48
49 void create_312_transform(float transform[4][4], float xt, float yt, float zt, float zr, float xr, float
    yr)
50 {
51     float degtopi = 3.1415928/180.0;
52     float zr_rad = zr * degtopi;
53     float xr_rad = xr * degtopi;
54     float yr_rad = yr * degtopi;
55
56     float cx = cos(xr_rad);
57     float cy = cos(yr_rad);
58     float cz = cos(zr_rad);
59     float sx = sin(xr_rad);
60     float sy = sin(yr_rad);
61     float sz = sin(zr_rad);
62
63     transform[0][0] = cy * sx * sz - cz * sy;
64     transform[0][1] = -cx * sz;
65     transform[0][2] = cy * cz + sx * sy * sz;
66     transform[0][3] = xt;
67
68     transform[1][0] = -cy * cz * sx - sy * sz;
69     transform[1][1] = cx * cz;
70     transform[1][2] = cy * sz - cz * sx * sy;
71     transform[1][3] = yt;
72
73     transform[2][0] = cx * cy;
74     transform[2][1] = sx;
75     transform[2][2] = cx * sy;
76     transform[2][3] = zt;
77
78     transform[3][0] = transform[3][1] = transform[3][2] = 0.0f;
79     transform[3][3] = 1.0f;
80 }

```

## 6.32 camera\_calibration.h

```

1 #ifndef CAMERA_CALIBRATION_H
2 #define CAMERA_CALIBRATION_H
3
4 struct CameraCalibration {
5     CameraCalibration(float principal_distance, float principal_x, float principal_y, float pixel_pitch) {
6         principal_distance_ = principal_distance;
7         principal_x_ = principal_x;
8         principal_y_ = principal_y;
9         pixel_pitch_ = pixel_pitch;
10    };
11    CameraCalibration() {
12        principal_distance_ = 0;
13        principal_x_ = 0;
14        principal_y_ = 0;
15        pixel_pitch_ = 0;
16    }
17    /*Camera Location & Calibration Locations ~ Right Hand Axis System (Positive Z Towards Oneself)*/
18    float principal_distance_; /* (mm) */
19    float principal_x_; /* (mm) */
20    float principal_y_; /* (mm) */
21    float pixel_pitch_; /* pixel size in mm (mm/pixel) */
22 };
23
24 #endif /* CAMERA_CALIBRATION_H */

```

## 6.33 cuda\_launch\_parameters.h

```

1 #ifndef CUDA_LAUNCH_PARAMETERS_H
2 #define CUDA_LAUNCH_PARAMETERS_H
3
4 const int threads_per_block = 256;
5 const int maximum_stride_size = 10000000;
6
7 #endif /*CUDA_LAUNCH_PARAMETERS_H*/

```

## 6.34 gpu\_toolbox.h

```

1  #ifndef GPU_TOOLBOX_H
2  #define GPU_TOOLBOX_H
3
4  /*CUDA Custom Registration Namespace (Compiling as DLL)*/
5  namespace gpu_cost_function {
6      /*This class is a toolbox for users looking to write their own
7      cost functions, and uses GPU computing. Users will be provided
8      with several resources (stored on the GPU) and functions
9      (computed on the GPU):
10     Resources (Stored on GPU Memory During Initialization of
11     CostFunctionToolboxGPU Class):
12     - Every image uploaded to JTA
13     - Edge detected version of every image uploaded to JTA
14     - Dilated version of every image uploaded to JTA
15     (dilation value is same as "Dilation" int parameter
16     in Cost Function chosen for stage. If such a parameter
17     does not exist dilation is 0 and this image is a copy
18     of the edge detected version).
19     - GPU model classes (one for each model). The GPU model
20     for the primary model will be stored separately
21     from the list of GPU models for the non-primary
22     models.
23     - GPU model class also includes information
24     about the model such as the name, file location,
25     model type, and STL style triangle information.
26     This last piece of information is stored on the GPU.
27     - Render Engine Class. As of 4/11/2018 this will have
28     to be modified to accept device pointers for the triangle
29     normals and vertices.
30
31
32 */
33     class CostFunctionToolboxGPU {
34     }
35 }
36
37 #endif /* GPU_TOOLBOX_H */

```

## 6.35 pixel\_grayscale\_colors.h

```

1  #pragma once
2
3  /*Uchar Colors*/
4  #define WHITE_PIXEL 255
5  #define BLACK_PIXEL 0
6  #define EDGE_PIXEL 100
7  #define DILATED_PIXEL 99

```

## 6.36 pose\_matrix.h

```

1  #pragma once
2
3  /*Render Engine Header for Pose Class*/
4  #include "gpu/render_engine.cuh"
5
6  /*Standard Library*/
7  #include <vector>
8  #include <string>
9
10 /*Class for Storing and Retrieving Pose linked to a unique Frame/Model Pair*/
11 class PoseMatrix {
12 public:
13     /*Blank Constructor/Destructor*/
14     __declspec(dllexport) PoseMatrix();
15     __declspec(dllexport) ~PoseMatrix();
16
17     /*Add New Model to Pose Matrix*/
18     __declspec(dllexport) void AddModel(std::vector<gpu_cost_function::Pose > , std::string model_name,
19     bool is_principal_model);
19
20     /*Get Model Pose (True if Successful, Else False) - Pose is Returned by Passing via reference*/
21     __declspec(dllexport) bool GetModelPose(std::string model_name, int frame_index,
22     gpu_cost_function::Pose* pose_container);
23
24     /*Get Principal Model Pose*/
25     __declspec(dllexport) bool GetModelPose(int frame_index, gpu_cost_function::Pose* pose_container);
26
27     /*Update Stored Pose for Principal Model at given frame*/

```



```

25     __declspec(dllexport) bool UpdatePrincipalModelPose(int frame_index, gpu_cost_function::Pose
pose_container);
26
27 private:
28     /*Principal Model Name*/
29     std::string principal_model_name_;
30
31     /*Principal Model Lookup Index*/
32     int principal_model_index_;
33
34     /*Vector of All Models (Order Implies Index)*/
35     std::vector<std::string> model_names_;
36
37     /*Vector of All Model's Vector of Frame Poses
38 Size of pose_matrix_ = # of models by # of frames*/
39     std::vector<std::vector<gpu_cost_function::Pose>> pose_matrix_;
40
41 };

```

## 6.37 about.h

```

1 #ifndef ABOUT_H
2 #define ABOUT_H
3
4 #include <qdialog.h>
5 #include "ui_About.h"
6
7 //About JTA Popup Header
8 class About : public QDialog
9 {
10     Q_OBJECT
11
12 public:
13     About(QWidget *parent = 0, Qt::WindowFlags flags = 0);
14     ~About();
15     void setVersion(int A, int B, int C); // Sets Version Number Label
16
17 private:
18     Ui::aboutJTA ui;
19
20 };
21
22 #endif // ABOUT_H

```

## 6.38 controls.h

```

1 #ifndef CONTROLS_H
2 #define CONTROLS_H
3
4 #include <qdialog.h>
5 #include "ui_controls.h"
6 #include <qgraphicsscene.h>
7 #include <qscrollbar.h>
8 #include <qgraphicsview.h>
9 #include <QGraphicsPixmapItem>
10 #include <qimage.h>
11 #include <qevent.h>
12
13 //Controls JTA Popup Header
14 class Controls : public QDialog
15 {
16     Q_OBJECT
17
18 public:
19     Controls(QWidget *parent = 0, Qt::WindowFlags flags = 0);
20     ~Controls();
21
22 private:
23     Ui::controls ui;
24     QGraphicsScene *center_scene;
25     QGraphicsView *center_graph;
26     QGraphicsPixmapItem *center_item;
27 };
28
29 #endif // CONTROLS_H

```

## 6.39 drr\_tool.h

```

1  #ifndef DRR_TOOL_H
2  #define DRR_TOOL_H
3
4  #include <qdialog.h>
5  #include "ui_drr_tool.h"
6
7  /*Standard Library*/
8  #include <vector>
9
10 /*VTK*/
11 #include <vtkInteractorStyleTrackballActor.h>
12 #include <vtkRenderer.h>
13 #include <vtkSTLReader.h>
14 #include <vtkSmartPointer.h>
15 #include <vtkRenderWindow.h>
16 #include <vtkActor.h>
17 #include <vtkPolyDataMapper.h>
18 #include <vtkCamera.h>
19 #include <vtkProperty.h>
20 #include <vtkAlgorithm.h>
21
22 /*Models*/
23 #include "core/model.h"
24
25 /*GPU Models*/
26 #include "gpu/gpu_model.cuh"
27
28 //About JTA Popup Header
29 class DRRTool : public QDialog
30 {
31     Q_OBJECT
32
33 public:
34     DRRTool(Model model, CameraCalibration calibration, double model_z_plane, QWidget *parent = 0,
35             Qt::WindowFlags flags = 0);
36     ~DRRTool();
37
38     /*Draw DRR*/
39     void DrawDRR();
40
41 public slots:
42     /*Threshold Changes*/
43     void on_minLowerSpinBox_valueChanged();
44     void on_maxLowerSpinBox_valueChanged();
45     void on_minUpperSpinBox_valueChanged();
46     void on_maxUpperSpinBox_valueChanged();
47     void on_minSlider_valueChanged();
48     void on_maxSlider_valueChanged();
49
50 private:
51     Ui::drrTool ui;
52
53     /*VTK*/
54     vtkSmartPointer<vtkRenderer> renderer_;
55
56     /*Actor and Mapper and Model (CPU and GPU)*/
57     vtkSmartPointer<vtkActor> actor_;
58     vtkSmartPointer<vtkPolyDataMapper> mapper_;
59     Model model_;
60     gpu_cost_function::GPUModel* gpu_model_;
61
62     /*Camera Stuff*/
63     float principal_distance_;
64     float pixel_pitch_;
65     float principal_y_;
66     CameraCalibration calibration_;
67
68     /*Array for Storing Device Image on Host*/
69     unsigned char* host_image_;
70
71     /*QImage for Converting Host to Viewable Image*/
72     QImage qt_host_image_;
73
74 };
75
76 #endif // DRR_TOOL_H

```

## 6.40 interactor.h

```

1  #ifndef INTERACTOR_H

```

```

2 #define INTERACTOR_H
3
4 #include <vtkObjectFactory.h>
5 #include <vtkInteractorStyleTrackballActor.h>
6 #include <vtkRendererCollection.h>
7 #include <vtkTextActor.h>
8 #include <vtkTextProperty.h>
9 #include <vtkActor2DCollection.h>
10 #include <vtkPicker.h>
11 #include <vtkPropPicker.h>
12 #include <vtkProp.h>
13 #include <qcursor.h>
14
15 /*Ref to QMainWindow*/
16 #include "gui/mainscreen.h"
17
18 //Calibration To Convert Pose
19 #include "core/calibration.h"
20 Calibration interactor_calibration;
21
22 //Speed of Movement
23 int speed = 1;
24 bool information = true;
25 bool interactor_camera_B = false; //Are we in Camera B?
26 bool middleDown = false; // Is CM button down?
27 bool leftDown = false; //Is LM button down?
28 bool rightDown = false; //Is RM button down
29 int rightDownY = 0; //Y Pixel when RM Clicked
30 double rightDownModelZ = 0; //Model's Z Translation when RM Clicked
31
32 class KeyPressInteractorStyle : public vtkInteractorStyleTrackballActor
33 {
34 public:
35     static KeyPressInteractorStyle* New();
36     vtkTypeMacro(KeyPressInteractorStyle, vtkInteractorStyleTrackballActor);
37
38     /*Pointer to Main Window*/
39     MainScreen* ms_;
40     void initialize_MainScreen(MainScreen* ms) {
41         ms_ = ms;
42     }
43
44     //Picked Function
45     bool ActivePick()
46     {
47         if (this->InteractionProp == NULL) return false;
48         else return true;
49     }
50
51     //KeyPress Turns Off Other Char Hotkeys
52     virtual void OnChar() {
53         vtkRenderWindowInteractor *rwi = this->Interactor;
54         std::string key = rwi->GetKeySym();
55         if (key == "Escape" || key == "escape" || key == "ESC" || key == "Esc" || key == "esc")
56         {
57             ms_->VTKEscapeSignal();
58         }
59     }
60
61     //Keypress Function
62     virtual void OnKeyPress()
63     {
64         // Get the keypress
65         vtkRenderWindowInteractor *rwi = this->Interactor;
66         if (this->InteractionProp == NULL)
67         {
68             std::string key = rwi->GetKeySym();
69
70             // Handle information toggle
71             if (key == "i" || key == "I")
72             {
73                 vtkTextActor* text =
74                 vtkTextActor::SafeDownCast(this->Interactor->GetRenderWindow()->GetRenderers()->GetFirstRenderer()->GetActors2D()->GetL
75                 if (information == true) { information = false; text->GetTextProperty()->SetOpacity(0.0);
76             }
77                 else { information = true; text->GetTextProperty()->SetOpacity(1.0); }
78
79                 this->Interactor->GetRenderWindow()->Render();
80                 return;
81             }
82
83             vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
84             std::string key = rwi->GetKeySym();
85             double* Position = actor->GetPosition();
86
87             //Shift Class

```

```

87         if (rwi->GetShiftKey())
88         {
89             //Handle Increase Request
90             if (key == "plus")
91                 if (speed < 20) speed++;
92
93             //Handle Decrease Request
94             if (key == "underscore")
95                 if (speed > 1) speed--;
96
97             // Handle an arrow key
98             if (key == "Up")
99             {
100                 actor->RotateX(speed);
101                 this->Interactor->GetRenderWindow()->Render();
102             }
103             // Handle an arrow key
104             if (key == "Down")
105             {
106                 actor->RotateX(-1 * speed);
107                 this->Interactor->GetRenderWindow()->Render();
108             }
109
110             // Handle an arrow key
111             if (key == "Left")
112             {
113                 actor->RotateY(-1 * speed);
114                 this->Interactor->GetRenderWindow()->Render();
115             }
116
117             // Handle an arrow key
118             if (key == "Right")
119             {
120                 actor->RotateY(speed);
121                 this->Interactor->GetRenderWindow()->Render();
122             }
123         }
124         //Control Class
125         else if (rwi->GetControlKey())
126         {
127             // Handle an arrow key
128             if (key == "Up")
129             {
130                 actor->SetPosition(Position[0], Position[1], Position[2] + speed);
131                 this->Interactor->GetRenderWindow()->Render();
132             }
133             // Handle an arrow key
134             if (key == "Down")
135             {
136                 actor->SetPosition(Position[0], Position[1], Position[2] - speed);
137                 this->Interactor->GetRenderWindow()->Render();
138             }
139
140             // Handle an arrow key
141             if (key == "Left")
142             {
143                 actor->RotateZ(-1 * speed);
144                 this->Interactor->GetRenderWindow()->Render();
145             }
146
147             // Handle an arrow key
148             if (key == "Right")
149             {
150                 actor->RotateZ(speed);
151                 this->Interactor->GetRenderWindow()->Render();
152             }
153         }
154         //Naked Class
155         else
156         {
157             //Handle Increase Request
158             if (key == "equal")
159                 if (speed < 20) speed++;
160
161             //Handle Decrease Request
162             if (key == "minus")
163                 if (speed > 1) speed--;
164
165             // Handle an arrow key
166             if (key == "Up")
167             {
168                 actor->SetPosition(Position[0], Position[1] + speed, Position[2]);
169                 this->Interactor->GetRenderWindow()->Render();
170             }
171             // Handle an arrow key
172             if (key == "Down")
173             {

```

```

174         actor->SetPosition(Position[0], Position[1] - speed, Position[2]);
175         this->Interactor->GetRenderWindow()->Render();
176     }
177
178     // Handle an arrow key
179     if (key == "Left")
180     {
181         actor->SetPosition(Position[0] - speed, Position[1], Position[2]);
182         this->Interactor->GetRenderWindow()->Render();
183     }
184
185     // Handle an arrow key
186     if (key == "Right")
187     {
188         actor->SetPosition(Position[0] + speed, Position[1], Position[2]);
189         this->Interactor->GetRenderWindow()->Render();
190     }
191
192     // Handle information toggle
193     if (key == "i" || key == "I")
194     {
195         if (information == true) information = false;
196         else information = true;
197     }
198
199     // Handle information toggle
200     if (key == "p" || key == "P")
201     {
202         if (!ms_->currently_optimizing_) {
203             ms_->VTKMakePrincipalSignal(actor);
204             return;
205         }
206     }
207
208 }
209
210 //Information Toggle
211 std::string infoText = "Location: <";
212 vtkTextActor* text =
213 vtkTextActor::SafeDownCast(this->Interactor->GetRenderWindow()->GetRenderers()->GetFirstRenderer()->GetActors2D()->GetL
214 if (information == true)
215 {
216     if (interactor_camera_B == false) {
217         infoText += std::to_string((long double)actor->GetPosition()[0]) + ", "
218         + std::to_string((long double)actor->GetPosition()[1]) + ", "
219         + std::to_string((long double)actor->GetPosition()[2]) + ">\nOrientation: <"
220         + std::to_string((long double)actor->GetOrientation()[0]) + ", "
221         + std::to_string((long double)actor->GetOrientation()[1]) + ", "
222         + std::to_string((long double)actor->GetOrientation()[2]) + ">\nKeyboard Speed: " +
223         std::to_string((int)speed);
224     }
225     else {
226         Point6D current_position_B = Point6D(actor->GetPosition()[0], actor->GetPosition()[1],
227 actor->GetPosition()[2],
228 actor->GetOrientation()[0], actor->GetOrientation()[1], actor->GetOrientation()[2]);
229 Point6D current_position_A =
230 interactor_calibration.convert_Pose_B_to_Pose_A(current_position_B);
231 infoText += std::to_string((long double)current_position_A.x) + ", "
232 + std::to_string((long double)current_position_A.y) + ", "
233 + std::to_string((long double)current_position_A.z) + ">\nOrientation: <"
234 + std::to_string((long double)current_position_A.xa) + ", "
235 + std::to_string((long double)current_position_A.ya) + ", "
236 + std::to_string((long double)current_position_A.za) + ">\nKeyboard Speed: " +
237 std::to_string((int)speed);
238 }
239 text->GetTextProperty()->SetOpacity(1.0);
240 text->GetTextProperty()->SetColor(actor->GetProperty()->GetColor());
241 }
242 else
243 text->GetTextProperty()->SetOpacity(0.0);
244 text->SetInput(infoText.c_str());
245 this->Interactor->GetRenderWindow()->Render();
246
247 //Forward events
248 vtkInteractorStyleTrackballActor::OnKeyPress();
249 }
250
251 //Left Mouse Down Function
252 virtual void OnLeftButtonDown()
253 {
254     leftDown = true;
255     // Forward Events
256     vtkInteractorStyleTrackballActor::OnLeftButtonDown();
257 }

```

```

256 //Right Mouse Down Function
257 virtual void OnRightButtonDown()
258 {
259     rightDown = true;
260     rightDownY = QCursor::pos().y();
261
262     // Forward Events
263     vtkInteractorStyleTrackballActor::OnRightButtonDown();
264
265     if (this->InteractionProp == NULL)
266         return;
267     vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
268     rightDownModelZ = actor->GetPosition()[2];
269 }
270
271 //Middle Mouse Down Funtion
272 virtual void OnMiddleButtonDown() {
273     middleDown = true;
274
275     // Forward Events
276     vtkInteractorStyleTrackballActor::OnMiddleButtonDown();
277 }
278
279 //Left Mouse Up Function
280 virtual void OnLeftButtonUp()
281 {
282     if (this->InteractionProp == NULL)
283         return;
284     vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
285
286     leftDown = false;
287     //Information Toggle
288     std::string infoText = "Location: <";
289     vtkTextActor* text =
290     vtkTextActor::SafeDownCast(this->Interactor->GetRenderWindow()->GetRenderers()->GetFirstRenderer()->GetActors2D()->GetL
291     {
292         if (information == true)
293         {
294             if (interactor_camera_B == false) {
295                 infoText += std::to_string((long double)actor->GetPosition()[0]) + ","
296                 + std::to_string((long double)actor->GetPosition()[1]) + ","
297                 + std::to_string((long double)actor->GetPosition()[2]) + ">\nOrientation: <"
298                 + std::to_string((long double)actor->GetOrientation()[0]) + ","
299                 + std::to_string((long double)actor->GetOrientation()[1]) + ","
300                 + std::to_string((long double)actor->GetOrientation()[2]) + ">\nKeyboard Speed: " +
301                 std::to_string((int)speed);
302             }
303             else {
304                 Point6D current_position_B = Point6D(actor->GetPosition()[0], actor->GetPosition()[1],
305                 actor->GetPosition()[2],
306                 actor->GetOrientation()[0], actor->GetOrientation()[1], actor->GetOrientation()[2]);
307                 Point6D current_position_A =
308                 interactor_calibration.convert_Pose_B_to_Pose_A(current_position_B);
309                 infoText += std::to_string((long double)current_position_A.x) + ","
310                 + std::to_string((long double)current_position_A.y) + ","
311                 + std::to_string((long double)current_position_A.z) + ">\nOrientation: <"
312                 + std::to_string((long double)current_position_A.xa) + ","
313                 + std::to_string((long double)current_position_A.ya) + ","
314                 + std::to_string((long double)current_position_A.za) + ">\nKeyboard Speed: " +
315                 std::to_string((int)speed);
316             }
317             text->GetTextProperty()->SetOpacity(1.0);
318             text->GetTextProperty()->SetColor(actor->GetProperty()->GetColor());
319         }
320         else
321         {
322             text->GetTextProperty()->SetOpacity(0.0);
323             text->SetInput(infoText.c_str());
324             this->Interactor->GetRenderWindow()->Render();
325             // Forward Events
326             vtkInteractorStyleTrackballActor::OnLeftButtonUp();
327         }
328     }
329
330 //Right Mouse Up Function
331 virtual void OnRightButtonUp()
332 {
333     rightDown = false;
334
335     // Forward Events
336     vtkInteractorStyleTrackballActor::OnRightButtonUp();
337 }
338
339 //Middle Mouse Up Function
340 virtual void OnMiddleButtonUp() {
341     middleDown = false;
342
343     //Forward Events
344     vtkInteractorStyleTrackballActor::OnMiddleButtonUp();
345 }

```

```

338     }
339
340     //Mouse Movement
341     virtual void OnMouseMove()
342     {
343         if (this->InteractionProp == NULL) return;
344         if (leftDown == true || rightDown == true || middleDown == true)
345         {
346             vtkActor *actor = vtkActor::SafeDownCast(this->InteractionProp);
347
348             //If Right Down and Not Left or MiddleScale The Z
349             if (!leftDown && !middleDown)
350             {
351                 double* Position = actor->GetPosition();
352                 actor->SetPosition(Position[0], Position[1], QCursor::pos().y() - rightDownY +
rightDownModelZ);
353             }
354
355             //Information Toggle
356             std::string infoText = "Location: <";
357             vtkTextActor* text =
vtkTextActor::SafeDownCast(this->Interactor->GetRenderWindow()->GetRenderers()->GetFirstRenderer()->GetActors2D()->GetL
358             if (information == true)
359             {
360                 if (interactor_camera_B == false) {
361                     infoText += std::to_string((long double)actor->GetPosition()[0]) + ", "
362                     + std::to_string((long double)actor->GetPosition()[1]) + ", "
363                     + std::to_string((long double)actor->GetPosition()[2]) + ">\nOrientation: <"
364                     + std::to_string((long double)actor->GetOrientation()[0]) + ", "
365                     + std::to_string((long double)actor->GetOrientation()[1]) + ", "
366                     + std::to_string((long double)actor->GetOrientation()[2]) + ">\nKeyboard Speed:
" + std::to_string((int)speed);
367                 }
368             }
369             else {
370                 Point6D current_position_B = Point6D(actor->GetPosition()[0],
actor->GetPosition()[1], actor->GetPosition()[2],
371                 actor->GetOrientation()[0], actor->GetOrientation()[1],
actor->GetOrientation()[2]);
372                 Point6D current_position_A =
interactor_calibration.convert_Pose_B_to_Pose_A(current_position_B);
373                 infoText += std::to_string((long double)current_position_A.x) + ", "
374                 + std::to_string((long double)current_position_A.y) + ", "
375                 + std::to_string((long double)current_position_A.z) + ">\nOrientation: <"
376                 + std::to_string((long double)current_position_A.xa) + ", "
377                 + std::to_string((long double)current_position_A.ya) + ", "
378                 + std::to_string((long double)current_position_A.za) + ">\nKeyboard Speed: " +
std::to_string((int)speed);
379             }
380             text->GetTextProperty()->SetOpacity(1.0);
381             text->GetTextProperty()->SetColor(actor->GetProperty()->GetColor());
382         }
383         else
384             text->GetTextProperty()->SetOpacity(0.0);
385         text->SetInput(infoText.c_str());
386         this->Interactor->GetRenderWindow()->Render();
387     }
388
389     // Forward Events
390     if (!rightDown)
391         vtkInteractorStyleTrackballActor::OnMouseMove();
392 }
393 };
394 vtkStandardNewMacro(KeyPressInteractorStyle);
395
396
397 class CameraInteractorStyle : public vtkInteractorStyleTrackballCamera
398 {
399 public:
400     static CameraInteractorStyle* New();
401     vtkTypeMacro(CameraInteractorStyle, vtkInteractorStyleTrackballCamera);
402
403     //KeyPress Turns Off Other Char Hotkeys
404     virtual void OnChar() {}
405 };
406 vtkStandardNewMacro(CameraInteractorStyle);
407
408 #endif /* INTERACTOR_H */

```

## 6.41 mainscreen.h

```

1 #ifndef MAINSCREEN_H
2 #define MAINSCREEN_H

```

```

3
4 /*Relevant QT Includes*/
5 #include <QtWidgets/MainWindow>
6 #include <qactiongroup.h>
7 #include "ui_mainscreen.h"
8 #include <memory.h>
9 /*Font*/
10 #include <qfont.h>
11
12 /*Key Event*/
13 #include <QKeyEvent>
14
15 #include "nfd/nfd.h"
16
17 /*Direct Data Structures*/
18 #include "core/data_structures_6D.h"
19
20 /*Custom Calibration Struct (Used in CUDA GPU METRICS)*/
21 #include "core/calibration.h"
22
23 /*VTK*/
24 #include <vtkRenderWindow.h>
25 #include <vtkProperty.h>
26 #include <vtkCamera.h>
27 #include <vtkSmartPointer.h>
28 #include <vtkImageData.h>
29 #include <vtkDataSetMapper.h>
30 #include <vtkActor.h>
31 #include <vtkRenderWindow.h>
32 #include <vtkRenderer.h>
33 #include <vtkRenderWindowInteractor.h>
34 #include <vtkVersion.h>
35 #include <vtkSTLReader.h>
36 #include <vtkImageImport.h>
37 #include <vtkPolyDataMapper.h>
38 #include <vtkRenderWindowInteractor.h>
39 #include <vtkInteractorStyleTrackballActor.h>
40 #include <vtkTextActor.h>
41 #include <vtkTextProperty.h>
42 #include <vtkAutoInit.h> // Added post migration to Banks' lab computer
43 #include <vtkInteractorStyleTrackballCamera.h> /*Alternate Camera*/
44
45 /*Frame and Model and Location Storage*/
46 #include "core/frame.h"
47 #include "core/model.h"
48 #include "core/location_storage.h"
49
50 /*Optimizer Settings*/
51 #include "core/optimizer_settings.h"
52
53 /*Optimizer Manager*/
54 #include "core/optimizer_manager.h"
55
56 /*Optimizer Settings Control Window*/
57 #include "gui/settings_control.h"
58
59 /*DRR Settings Control Window*/
60 #include "drr_tool.h"
61
62 /* Symmetry Trap Analysis Window*/
63 #include "gui/sym_trap.h"
64
65 /*Cost Function Library*/
66 #include "cost_functions/CostFunctionManager.h"
67
68 /*CostFunctionTools*/
69 #include "camera_calibration.h"
70
71 /*machine_learning_tools*/
72 #include "core/machine_learning_tools.h"
73
74 #include "nfd/nfd.h"
75
76 #include "gui/viewer.h"
77
78 class MainScreen : public QMainWindow
79 {
80     Q_OBJECT
81
82 public:
83     MainScreen(QWidget* parent = 0);
84     ~MainScreen();
85
86     /*Escape Signal from VTK to stop optimizer*/
87     void VTKEscapeSignal();
88
89     /*Make Selected Actor Principal from VTK*/

```



```

90     void VTKMakePrincipalSignal(vtkActor* new_principal_actor);
91
92     /*Bool to see if currently optimizing*/
93     bool currently_optimizing_;
94
95
96
97
98 Q_SIGNALS:
99     /*Update Whether To Write TO Text Display*/
100     void UpdateDisplayText(bool);
101     /*Stop Optimizer*/
102     void StopOptimizer();
103
104     // [SYM TRAP] Send out optimizer time remaining
105     void UpdateTimeRemaining(int);
106
107 private:
108     Ui::MainScreenClass ui;
109
110     float start_time;
111
112     /*GUI FUNCTIONS*/
113     /*Arrange Layout (Do this in code so scales across different DPI monitors and handles weird fonts)*/
114     void ArrangeMainScreenLayout(QFont application_font);
115
116     /*Private Variables*/
117     /*Original Sizes After Construction for Main Screen List Widgets, their Group Boxes and QVTK
Widget*/
118     int image_list_widget_starting_height_;
119     int image_selection_box_starting_height_;
120     int model_list_widget_starting_height_;
121     int model_selection_box_starting_height_;
122     int qvtk_widget_starting_height_;
123     int qvtk_widget_starting_width_;
124
125     /*Monoplane and Biplane Calibration Viewport Files*/
126     Calibration calibration_file_; /*Used in monoplane and biplane*/
127
128     /*Variables Indicating Calibration Status for Mono and Biplane*/
129     bool calibrated_for_monoplane_viewport_;
130     bool calibrated_for_biplane_viewport_;
131
132     /*VTK Variables*/
133     std::vector<vtkSmartPointer<vtkActor>> model_actor_list;
134     std::vector<vtkSmartPointer<vtkPolyDataMapper>> model_mapper_list;
135     vtkSmartPointer<vtkRenderer> renderer;
136     vtkSmartPointer<vtkImageData> current_background;
137     vtkSmartPointer<vtkSTLReader> stl_reader;
138     vtkSmartPointer<vtkDataSetMapper> image_mapper;
139     vtkSmartPointer<vtkActor> actor_image;
140     vtkSmartPointer<vtkTextActor> actor_text;
141     vtkSmartPointer<vtkImageImport> importer;
142     vtkSmartPointer<vtkInteractorStyleTrackballCamera> camera_style_interactor;
143
144
145     std::shared_ptr<viewer> vw = std::make_shared<viewer>();
146
147
148     /*View Menu Radio Button Container*/
149     QActionGroup* alignmentGroup, * alignmentGroupSegment;
150
151     /*Frame/Model Containers*/
152     std::vector<Frame> loaded_frames;
153     std::vector<Frame> loaded_frames_B; /*If Biplane mode, need second group of loaded frames for camera
B*/
154     std::vector<Model> loaded_models;
155
156     /*Location Storage Class*/
157     LocationStorage model_locations_;
158
159     /*Index of Previously Selected Frame/Models*/
160     int previous_frame_index_;
161     QModelIndexList previous_model_indices_;
162
163     /*Save the Pose From The Last Selected Frame*/
164     void SaveLastPose();
165
166     /*Optimizer Settings That Must Be Set in Constructor and Changed on OSettings Update */
167     OptimizerSettings optimizer_settings_;
168
169     /*Copy of the Above Only Used While Optimizing to Display Output*/
170     OptimizerSettings display_optimizer_settings_;
171
172     /*Cost Function Managers (from JTA Cost Function Library) for each stage of DIRECT-JTA Optimizer*/
173     jta_cost_function::CostFunctionManager trunk_manager_;
174     jta_cost_function::CostFunctionManager branch_manager_;

```

```

175     jta_cost_function::CostFunctionManager leaf_manager_; // For extra Z-translation usually (esp. when
monoplane)
176
177     /*Function That Saves Dilation as 0 if No Trunk Manager has a Dilation Int Parameter,
178 else saves all the Dilation Images for Each Frame as the Dilation Constant*/
179     void UpdateDilationFrames();
180
181
182     /*Optimization Function: Packages Off The Optimization process in
183 a new thread*/
184
185     /*Launch Optimizer*/
186     void LaunchOptimizer(QString directive); //Directive Says whether it is Optimize Single, From, All,
or Each (or Sym_Trap)
187
188     /*Optimizer Thread and Manager*/
189     QThread* optimizer_thread;
190     OptimizerManager *optimizer_manager;
191
192     /*Disable and Enable MainScreen During and After Optimization*/
193     void DisableAll();
194     void EnableAll();
195
196     /*Function That Loads Settings from Registry or (If First Time Loading
197 Saves Default Settings*/
198     void LoadSettingsBetweenSessions();
199
200     /*Mat to Vtk*/
201     void matToVTK(cv::Mat Input, vtkSmartPointer<vtkImageData> Output);
202
203     /*Optimizer Window Control*/
204     SettingsControl* settings_control;
205
206     /*Sym Trap Window*/
207     sym_trap* sym_trap_control;
208
209     /*Calculate Viewing Angle (Accounts for Offsets)*/
210     double CalculateViewingAngle(int width, int height, bool CameraA);
211
212     /*Helper Function To Segment And Update Frames According to Model File*/
213     void segmentHelperFunction(std::string pt_model_location, unsigned int input_width, unsigned int
input_height);
214
215     // Helper function for sym_trap to get information about the current pose
216     Point6D copy_current_pose();
217     bool sym_trap_running;
218
219 public Q_SLOTS:
220
221     // Call Optimizer Launch
222     void optimizer_launch_slot();
223
224     /*Load Buttons*/
225     void on_load_calibration_button_clicked(); /*Load Calibration Clicked*/
226     void on_load_image_button_clicked(); /*Load Images*/
227     void on_load_model_button_clicked(); /*Load Models*/
228
229
230     /*Biplane View Button (Monoplane is Biplane A, Biplans is Biplane B*/
231     void on_camera_A_radio_button_clicked();
232     void on_camera_B_radio_button_clicked();
233
234     /*List Widgets*/
235     void on_image_list_widget_itemSelectionChanged(); /*Image List Widget Changed*/
236     void on_model_list_widget_itemSelectionChanged(); /*Model List Widget Changed*/
237
238     /*Multiple Selection For Models Radio buttons*/
239     void on_single_model_radio_button_clicked();
240     void on_multiple_model_radio_button_clicked();
241
242     /*Radio Buttons*/
243     /*Image Radio Buttons*/
244     void on_original_image_radio_button_clicked();
245     void on_inverted_image_radio_button_clicked();
246     void on_edges_image_radio_button_clicked();
247     void on_dilation_image_radio_button_clicked();
248     /*Model Radio Buttons*/
249     void on_original_model_radio_button_clicked();
250     void on_solid_model_radio_button_clicked();
251     void on_transparent_model_radio_button_clicked();
252     void on_wireframe_model_radio_button_clicked();
253
254     /*Edge Buttons*/
255     void on_aperture_spin_box_valueChanged();
256     void on_low_threshold_slider_valueChanged();
257     void on_high_threshold_slider_valueChanged();
258     void on_apply_all_edge_button_clicked();

```

```

259 void on_reset_edge_button_clicked();
260
261 /*MenuBar*/
262 void on_actionSave_Pose_triggered();
263 void on_actionSave_Kinematics_triggered();
264 void on_actionLoad_Pose_triggered();
265 void on_actionLoad_Kinematics_triggered();
266 void on_actionAbout_JointTrack_Auto_triggered();
267 void on_actionControls_triggered();
268 void on_actionStop_Optimizer_triggered();
269 void on_actionOptimizer_Settings_triggered();
270 void on_actionDRR_Settings_triggered();
271 void on_actionReset_View_triggered();
272 void on_actionReset_Normal_Up_triggered();
273 void on_actionModel_Interaction_Mode_triggered();
274 void on_actionCamera_Interaction_Mode_triggered();
275 void on_actionSegment_FemHR_triggered();
276 void on_actionSegment_TibHR_triggered();
277 void on_actionReset_Remove_All_Segmentation_triggered();
278 void on_actionEstimate_Femoral_Implant_s_triggered();
279 void on_actionEstimate_Tibial_Implant_s_triggered();
280
281 void on_actionNFD_Pose_Estimate_triggered();
282
283 void on_actionCopy_Next_Pose_triggered();
284 void on_actionCopy_Previous_Pose_triggered();
285
286 void on_actionLaunch_Tool_triggered();
287 void on_actionAmbiguous_Pose_Processing_triggered();
288 /*Optimization Buttons*/
289 void on_optimize_button_clicked();
290 void on_optimize_all_button_clicked();
291 void on_optimize_each_button_clicked();
292 void on_optimize_from_button_clicked();
293 void on_actionOptimize_Backward_triggered();
294
295 /*OPTIMIZATION SLOTS*/
296 /*Update Blue Current Optimum*/
297 void onUpdateOptimum(double, double, double, double, double, double, double, unsigned int);
298 /*Finished Optimizing Frame, Send Optimum to MainScreen*/
299 void onOptimizedFrame(double, double, double, double, double, double, double, bool, unsigned int, bool,
QString);
300 /*Uh oh There was an Error. String contains the message
301 */
302 void onOptimizerError(QString error_message);
303 /*Update Display with Speed, Cost Function Calls, Current Minimum*/
304 void onUpdateDisplay(double, int, double, unsigned int);
305 /*Update Dilation Background if Radio Button is on Dilation and Moving Between Trunks and
Branches*/
306 void onUpdateDilationBackground();
307 void updateOrientationSymTrap_MS(double, double, double, double, double, double, double);
308
309 /*On Optimizer Control Windows Save Setting*/
310 void onSaveSettings(OptimizerSettings, jta_cost_function::CostFunctionManager,
jta_cost_function::CostFunctionManager, jta_cost_function::CostFunctionManager);
311
312 protected:
313 void resizeEvent(QResizeEvent* event);
314 void keyPressEvent(QKeyEvent* event);
315 };
316
317 #endif /* MAINSCREEN_H */

```

## 6.42 settings\_control.h

```

1 #ifndef SETTINGS_CONTROL_H
2 #define SETTINGS_CONTROL_H
3
4 #include <qdialog.h>
5 #include "ui_settings_control.h"
6
7 /*Optimizer Settings Class*/
8 #include "core/optimizer_settings.h"
9
10 /*JTA Cost Function Class*/
11 #include "cost_functions/CostFunctionManager.h"
12
13 //About OptimizerSettings Popup Header
14 class SettingsControl : public QDialog
15 {
16     Q_OBJECT
17
18 public:

```

```

19     SettingsControl(QWidget *parent = 0, Qt::WindowFlags flags = 0);
20     ~SettingsControl();
21
22     /*Load Optimizer Settings from Main Window*/
23     void LoadSettings(jta_cost_function::CostFunctionManager sc_trunk_manager,
24                     jta_cost_function::CostFunctionManager sc_branch_manager,
25                     jta_cost_function::CostFunctionManager sc_leaf_manager,
26                     OptimizerSettings opt_settings);
27
28 private:
29     Ui::settings_control ui;
30
31     /*Local to Settings Control Cost Function Managers*/
32     jta_cost_function::CostFunctionManager sc_trunk_manager_;
33     jta_cost_function::CostFunctionManager sc_branch_manager_;
34     jta_cost_function::CostFunctionManager sc_leaf_manager_; // For extra Z-translation usually (esp.
    when monoplane)
35
36     /*Optimizer Settings for Everything but the Cost Function Stuff*/
37     OptimizerSettings opt_settings_;
38
39 public slots:
40
41     /*Save Button*/
42     void on_save_button_clicked();
43
44     /*Reset Button*/
45     void on_reset_button_clicked();
46
47     /*Cancel Button*/
48     void on_cancel_button_clicked();
49
50     /*Radio buttons for stage*/
51     void on_trunk_radioButton_clicked();
52     void on_branch_radioButton_clicked();
53     void on_leaf_radioButton_clicked();
54
55     /*List Widgets Changed*/
56     void on_cost_function_listWidget_itemSelectionChanged();
57     void on_cost_function_parameters_listWidget_itemSelectionChanged();
58
59     /*Optimizer Settings Buttons Toggled*/
60     void on_stage_enabled_checkBox_clicked();
61     void on_budget_spinBox_valueChanged();
62     void on_x_translation_spinBox_valueChanged();
63     void on_y_translation_spinBox_valueChanged();
64     void on_z_translation_spinBox_valueChanged();
65     void on_x_rotation_spinBox_valueChanged();
66     void on_y_rotation_spinBox_valueChanged();
67     void on_z_rotation_spinBox_valueChanged();
68     void on_branch_count_spinBox_valueChanged();
69     void on_double_parameter_spinBox_valueChanged();
70     void on_int_parameter_spinBox_valueChanged();
71     void on_bool_parameter_true_radioButton_clicked();
72     void on_bool_parameter_false_radioButton_clicked();
73
74
75
76 signals:
77     /*Saves the three Cost Function Manager Settings and the Optimizer Settings to:
78 - the registry
79 - their local class versions on the main window GUI*/
80     void SaveSettings(OptimizerSettings,
81                     jta_cost_function::CostFunctionManager, jta_cost_function::CostFunctionManager,
82                     jta_cost_function::CostFunctionManager); //
83     /*Close Window*/
84     void Done(); //
85
86 #endif /* SETTINGS_CONTROL_H */

```

## 6.43 sym\_trap.h

```

1 #pragma once
2 #include "ui_sym_trap.h"
3 #include "cost_functions/CostFunctionManager.h"
4 #include "core/data_structures_6D.h"
5
6 #include <qdialog.h>
7 #include <QFile>
8 #include <QTextStream>
9 #include <QtWidgets/qfiledialog.h>
10 #include <QMessageBox>

```

```

11 // #include "optimizer_manager.h"
12
13
14 #include <QVTKWidget.h>
15
16 #include <vtkImageData.h>
17 #include <vtkTextProperty.h>
18 #include <vtkAxis.h>
19 #include <vtkAxisActor2D.h>
20 #include <vtkChartXY.h>
21 #include <vtkContextScene.h>
22 #include <vtkContextView.h>
23 #include <vtkCubeAxesActor2D.h>
24 #include <vtkDataSetMapper.h>
25 #include <vtkFloatArray.h>
26 #include <vtkInteractorStyleTrackball.h>
27 #include <vtkInteractorStyleTrackballCamera.h>
28 #include <vtkNamedColors.h>
29 #include <vtkNew.h>
30 #include <vtkPlotPoints.h>
31 #include <vtkPNGWriter.h>
32 #include <vtkPointData.h>
33 #include <vtkPoints.h>
34 #include <vtkPolyData.h>
35 #include <vtkPolyDataMapper.h>
36 #include <vtkProperty.h>
37 #include <vtkRenderer.h>
38 #include <vtkRenderWindow.h>
39 #include <vtkRenderWindowInteractor.h>
40 #include <vtkSimplePointsReader.h>
41 #include <vtkSmartPointer.h>
42 #include <vtkTable.h>
43 #include <vtkWarpScalar.h>
44
45 #include <cmath>
46 #include <vector>
47 #include <iostream>
48 #include <sstream>
49
50
51 class sym_trap : public QDialog
52 {
53     Q_OBJECT
54
55 public:
56     sym_trap(QWidget* parent = 0, Qt::WindowFlags flags = 0);
57     ~sym_trap();
58     //OptimizerManager* sym_trap_optimizer = new OptimizerManager();
59
60     //Point6D pose;
61
62     Point6D compute_mirror_pose(Point6D point);
63     static void matmult4(float ans[4][4], float matrix1[4][4], float matrix2[4][4]);
64     static void matmult3(float ans[3][3], const float matrix1[3][3], const float matrix2[3][3]);
65     static void invert_transform(float result[4][4], const float tran[4][4]);
66     static void equivalent_axis_angle_rotation(float rot[3][3], const float m[3], const float angle);
67     static void cross_product(float CP[3], const float v1[3], const float v2[3]);
68     static void dot_product(float& result, const float vector1[3], const float vector2[3]);
69     static void rotation_matrix(float R[3][3], Point6D pose);
70     static void create_3l2_transform(float transform[4][4], Point6D pose);
71     static void getRotations3l2(float& xr, float& yr, float& zr, const float Rot[3][3]);
72
73     static void copy_matrix_by_value(float(&new_matrix)[3][3], const float(&old_matrix)[3][3]);
74     void create_vector_of_poses(std::vector<Point6D>& pose_list, Point6D pose);
75
76     template<typename T>
77     std::vector<double> static linspace(T start_in, T end_in, int num_in);
78
79     int getIterCount();
80
81     //void set_pose(Point6D desired_pose);
82     Ui::symTrap ui;
83
84 public Q_SLOTS:
85     double onCostFuncAtPoint(double result);
86     void graphResults();
87     void graphResults2D();
88     void setIterCount(int n);
89     void saveData();
90     void loadData();
91     void savePlot();
92
93 private:
94     std::vector<Point6D> search_space;
95     QVTKWidget* plot_widget;
96     int iter_count;

```

```

98
99
100 signals:
101     void Done();
102 };

```

## 6.44 viewer.h

```

1 #ifndef VIEWER_H
2 #define VIEWER_H
3
4 #pragma once
5 #include <vector>
6
7 #include <vtkRenderWindow.h>
8 #include <vtkProperty.h>
9 #include <vtkCamera.h>
10 #include <vtkSmartPointer.h>
11 #include <vtkImageData.h>
12 #include <vtkDataSetMapper.h>
13 #include <vtkActor.h>
14 #include <vtkRenderWindow.h>
15 #include <vtkRenderer.h>
16 #include <vtkRenderWindowInteractor.h>
17 #include <vtkVersion.h>
18 #include <vtkSTLReader.h>
19 #include <vtkImageImport.h>
20 #include <vtkPolyDataMapper.h>
21 #include <vtkRenderWindowInteractor.h>
22 #include <vtkInteractorStyleTrackballActor.h>
23 #include <vtkTextActor.h>
24 #include <vtkTextProperty.h>
25 #include <vtkAutoInit.h> // Added post migration to Banks' lab computer
26 #include <vtkInteractorStyleTrackballCamera.h> /*Alternate Camera*/
27 #include <iostream>
28
29 class viewer
30 {
31 public:
32     viewer();
33     ~viewer();
34
35     void initialize_vtk_pointers();
36     void initialize_vtk_mappers();
37     void load_render_window(vtkSmartPointer<vtkRenderWindow> in);
38     void initialize_vtk_renderers();
39     vtkSmartPointer<vtkRenderer> get_renderer();
40     vtkSmartPointer<vtkActor> get_actor_image();
41     vtkSmartPointer<vtkImageData> get_current_background();
42     vtkSmartPointer<vtkSTLReader> get_stl_reader();
43     vtkSmartPointer<vtkDataSetMapper> get_image_mapper();
44
45 private:
46     std::vector<vtkSmartPointer<vtkActor>> model_actor_list_;
47     std::vector<vtkSmartPointer<vtkPolyDataMapper>> model_mapper_list_;
48     vtkSmartPointer<vtkRenderer> background_renderer_;
49     vtkSmartPointer<vtkImageData> current_background_;
50     vtkSmartPointer<vtkSTLReader> stl_reader_;
51     vtkSmartPointer<vtkDataSetMapper> image_mapper_;
52     vtkSmartPointer<vtkActor> actor_image_;
53     vtkSmartPointer<vtkTextActor> actor_text_;
54     vtkSmartPointer<vtkImageImport> importer_;
55     vtkSmartPointer<vtkRenderWindow> qvtk_render_window_;
56     vtkSmartPointer<vtkCamera> my_image_camera_;
57     vtkSmartPointer<vtkCamera> my_model_camera_;
58
59     bool initialized_pointers_;
60 };
61
62 #endif

```

## 6.45 nfd.h

```

1 #pragma once
2
3 #include <iostream>

```

```

4 #include <QString>
5 #include "core/model.h"
6 #include "core/frame.h"
7 #include "core/calibration.h"
8
9 #include <gpu_model.cuh>
10 #include <gpu_intensity_frame.cuh>
11 #include <gpu_edge_frame.cuh>
12 #include <gpu_dilated_frame.cuh>
13 #include <gpu_metrics.cuh>
14 #include <vector>
15 #include <QModelIndex>
16 #include "cufft.h"
17 #include <opencv2/imgproc.hpp>
18
19 #include "nfd_instance.h"
20 #include "nfd_library.h"
21
22 using namespace gpu_cost_function;
23
24 class JTML_NFD {
25
26 public:
27     JTML_NFD();
28     ~JTML_NFD();
29     bool Initialize(
30         Calibration cal_file,
31         std::vector<Model> model_list,
32         std::vector<Frame> frames_list,
33         QModelIndexList selected_models,
34         unsigned int primary_model_index,
35         QString error_message
36     );
37
38     void Run();
39
40 private:
41     bool successful_initialization_;
42     Calibration calibration_;
43     std::vector<Frame> frames_;
44
45     GPUModel* gpu_principal_model_;
46
47     Model primary_model_;
48     std::vector<Model> all_models_;
49
50
51 };

```

## 6.46 nfd\_instance.h

```

1 #pragma once
2 #include<complex>
3 #include<vector>
4 #include<array>
5 #include "gpu/render_engine.cuh"
6 #include "gpu/gpu_model.cuh"
7 #include <opencv2/core/mat.hpp>
8 #include "opencv2/imgcodecs.hpp"
9 #include "opencv2/highgui.hpp"
10 #include "opencv2/imgproc.hpp"
11 #include "alglib/interpolation.h"
12
13 using namespace gpu_cost_function;
14 using namespace alglib;
15 /*
16 This class is beeing used to store a single instance of the NFD library data.
17
18 A single instance is hereby going to refer to a single projection geometry (meaning only a single x/y
19 rotation),
20 along with all the other data that might be associated with that. Multiple normalization coefficients
21 are going
22 to be defined for any one instance (as you see in the Banks paper).
23
24 The nfd_library class is going to be used to store a library of all the instances that we create for a
25 single projection,
26 with some extra metadata surrounding it.
27
28 The main nfd function is going to manage all the initialization of the different models, as well as
29 populating each of
30 the respective classes with the values that come from the different image projections.
31
32 */

```

```

29 struct vec2d {
30     float x;
31     float y;
32     void set_x(float xval) {
33         x = xval;
34     }
35     void set_y(float yval) {
36         y = yval;
37     }
38 };
39
40 class nfd_instance
41 {
42 public:
43
44     /*Constructor takes in the pose and creates the instance - will eventually also do the projections*/
45     nfd_instance(GPUModel &gpu_mod, float xt, float yt, float zt, float xr, float yr, float zr);
46     ~nfd_instance();
47
48     void print_contour_points();
49     void print_raw_points();
50
51     /*Set the pose of the instance - 312 rotation order when projected*/
52
53 private:
54
55     std::complex<float> centroid_;
56     std::vector<std::complex<float>> fourier_coefficients_[128];
57     float magnitude_;
58     std::vector<float> angle_;
59     float rot_x_;
60     float rot_y_;
61     Pose instance_pose_;
62     std::vector<std::vector<int>> contour_pts_;
63     //std::vector<int[2]> cntr_pts_;
64     real_2d_array contour_points_raw_;
65     std::vector<double> y_points_resampled_;
66     std::vector<double> x_points_resampled_;
67     pspline2interpolant contour_spline_;
68     int sz_;
69     void get_contour_points(cv::Mat img);
70
71
72 };

```

## 6.47 nfd\_library.h

```

1 #pragma once
2 #include <vector>
3 #include "nfd_instance.h"
4 #include "nfd.h"
5 #include <array>
6 #include "gpu/gpu_model.cuh"
7
8 /*
9 This NFD library is going to store a vector of each of the NFD instances at a given pose, as well as some
    metadata
10 about the overall scope of the library (rotation parameters and ranges, etc).
11 */
12 class nfd_library
13 {
14 public:
15     nfd_library(
16         Calibration cal_file,
17         GPUModel &gpu_model,
18         int x_range,
19         int y_range,
20         float x_inc,
21         float y_inc
22     );
23     ~nfd_library();
24     std::vector<nfd_instance> get_library();
25
26     void create_nfd_library();
27
28 private:
29     int x_range_;
30     int y_range_;
31     float x_inc_;
32     float y_inc_;
33     GPUModel* gpu_mod;
34     std::vector<std::array<float, 2>> rot_indices_;
35     void create_rot_indices(int x_range, int y_range, float x_inc, float y_inc);

```



```
36
37     std::vector<nfd_instance> library_;
38     Calibration calibration_;
39
40     nfd_instance testing_projection();
41
42 };
43
```





on\_actionNFD\_Pose\_Estimate\_triggered  
    MainScreen, [20](#)  
OptimizerManager, [22](#)  
OptimizerSettings, [23](#)  
  
Point6D, [25](#)  
PoseMatrix, [25](#)  
  
SettingsControl, [26](#)  
sym\_trap, [27](#)  
  
tibial\_pose\_selector  
    ambiguous\_pose\_processing.h, [32](#)  
  
varus\_valgus\_calculation  
    ambiguous\_pose\_processing.h, [32](#)  
vec2d, [28](#)  
Vect\_3, [28](#)  
viewer, [29](#)