Object Design and Integration Overview

SteamVR™ Tracking

Introduction

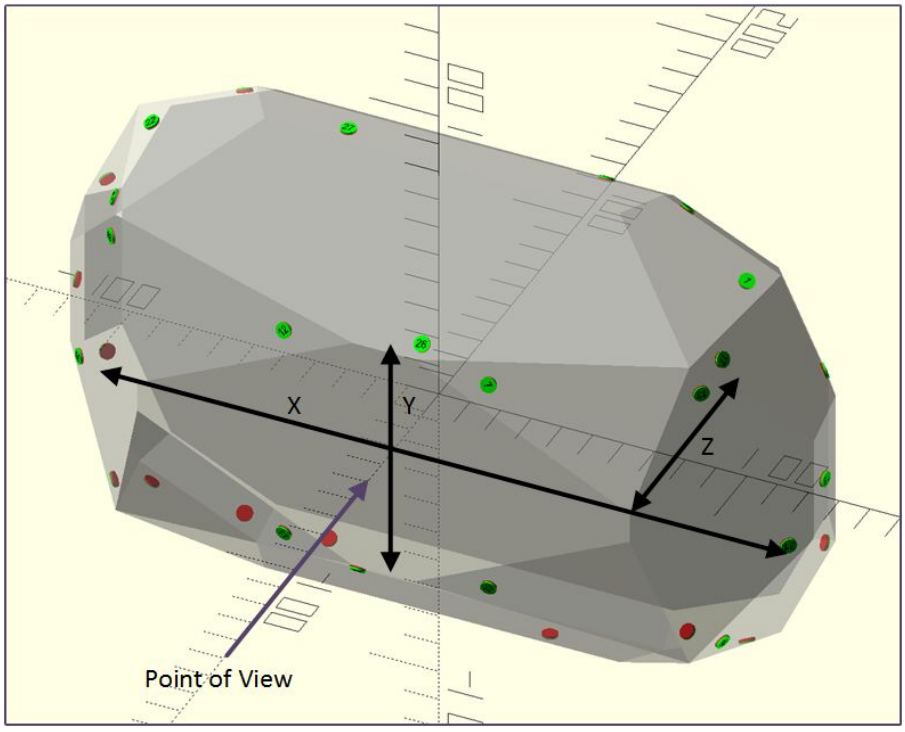
This document outlines the process of designing a tracked object for SteamVR™ Tracking. Each of the topics referenced here is covered in more detail in separate documents. However, this document provides a roadmap to guide designers from initial concept through integration with SteamVR™.

Designing for Sensor Placement

Defining the shape of the tracked object is not only the first step in the process, but one of the most important to tracking performance. When developing a product concept, industrial designers are often involved early in the process. Concerned about the ergonomics and look of a product, they have the user experience and aesthetics at heart. Industrial design for tracked objects adds another challenge, because the size, shape, and surface finish of objects has a direct impact on the tracking performance.

The surface of the object holds the sensors that receive the IR reference signals from the base station. It is important to remember that the sensors have an effective viewing angle of ±60°. The shape of the object and placement of sensors must ensure that the object meets the following criteria for high performance tracking:

1. Four sensors are visible to the base station and they are not coplanar. One of the sensors must be at least 8mm outside the plane created by the other three.
2. The distance between sensors (baseline) is maximized.
3. Sufficient baseline is present in all three axes.



Remember, from the perspective of the base station, sensor visibility and baseline change with every pose. Ideally, four visible non-coplanar sensors and three axes of sufficient baseline would exist at every orientation. However, it is not realistic to expect an object to have ideal visibility and baseline in every pose. Successful object design requires making tradeoffs to maximize tracking performance in the most common use cases.

It is very important for industrial designers to understand the basic rules of sensor placement to inform their design process. Some common design features, such as flat surfaces and right angles may have a negative impact on tracking. However, if industrial designers take the constraints of sensor placement as a seed for their design inspiration, they are likely to envision radically different concepts that are not only visually compelling but also track very well. See the document **Object Shape and Sensor Placement** for a more complete treatment of the topic.

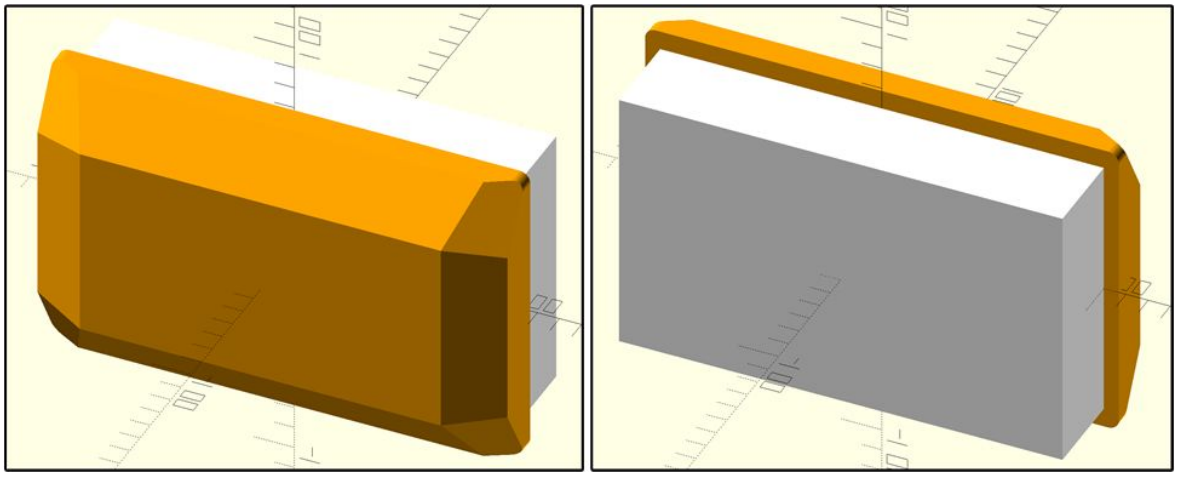
Industrial design often involves the recommendation of materials and surface finishes that define the look and feel of the object while protecting sensors from damage. However, covering sensors directly impacts tracking performance. In addition to sensor placement, it is important for industrial designers and mechanical engineers to understand best practices described in the **Sensor Covering** document.

Once the shape of an object is defined, it is important to verify that sensors may be placed on the surface to achieve acceptable tracking performance.

Generating Sensor Placement

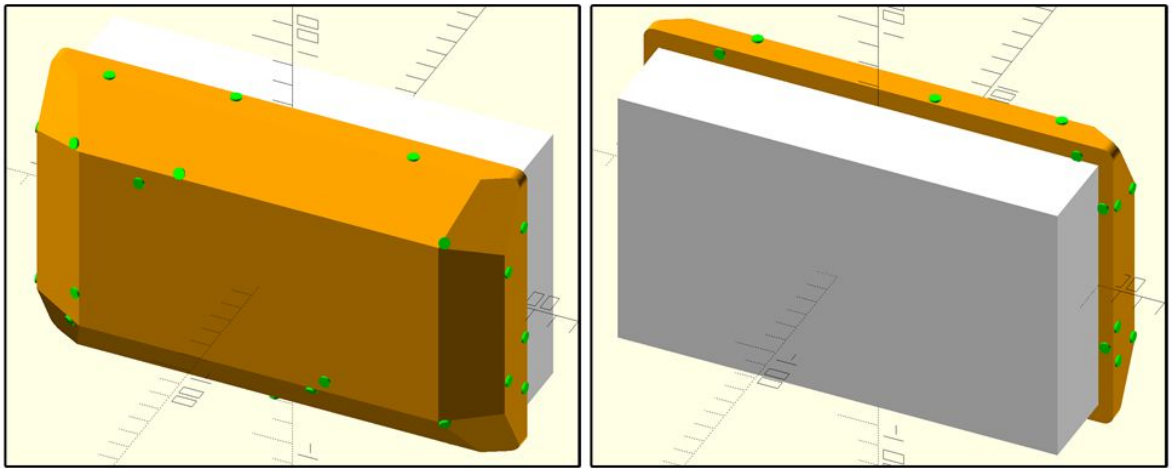
Valve developed a software tool that automatically places sensors on an arbitrary shape and optimizes the placement for the best tracking performance. The software tool hmd_designer is freely available to all licensees and includes many useful features for object design. One of the most useful, is its ability to generate sensor placement on the surface of an object. Hmd_designer accepts a 3D model in STL format. Follow these rules to prepare the STL file in a way that helps hmd_designer generate sensor placement effectively and efficiently:

1. Use a solid shape, not a hollow shell. Otherwise, the tool may waste time placing sensors inside the hollow shape, only to move them to the outside when it cannot find an interior position exposed to a base station.
2. Create STL files that represent parts of your object which cannot have sensors, such as handles or mounting points. The software will account for the “shadows” cast by these components where they block light arriving from the base station.
3. Create STL files that cover any surfaces on the object that should not hold sensors.



The software places sensors on the surface and optimizes their positions to maximize tracking performance. At first, allow the software to place all 32 sensors. If the shape does not yield good tracking results with 32 sensors, it will never be improved by decreasing the number of sensors. Once the sensor placement is generated, the tool outputs three files:

1. A JSON file, which describes the positions and orientations of each sensor. This file is part of the configuration which will be uploaded to the device.
2. A 3D model file in OpenSCAD format, which displays the object and all the sensors placed on the device.
3. A text file containing tracking simulation data, which may be displayed on 2D or 3D plots.

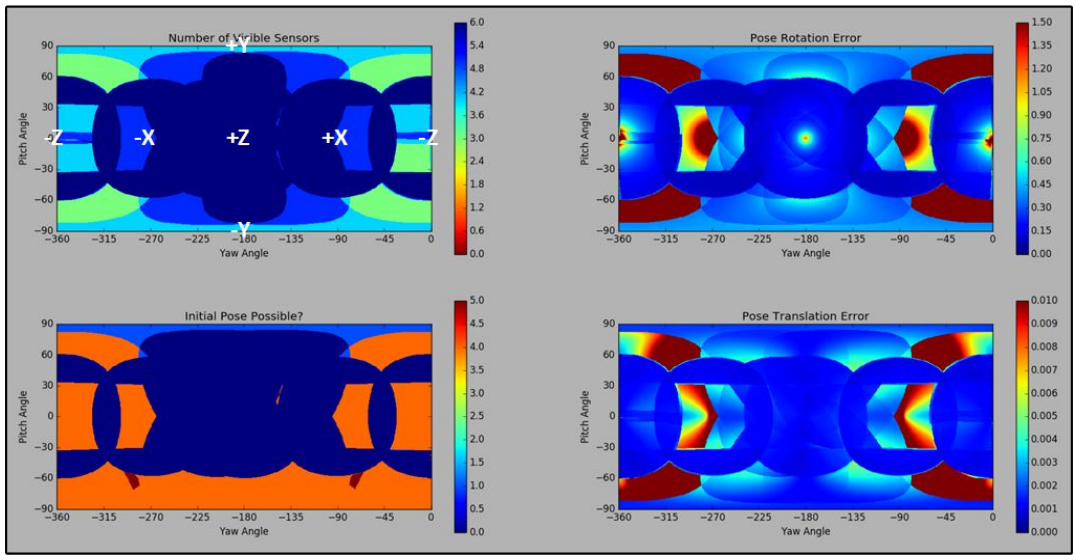


See “Generating Sensor Placement” in the **Simulation User Manual** for greater detail. Once sensor locations are generated and simulated, it is important to review the simulation output to identify areas where sensor coverage is not adequate, and update the shape to improve tracking quality.

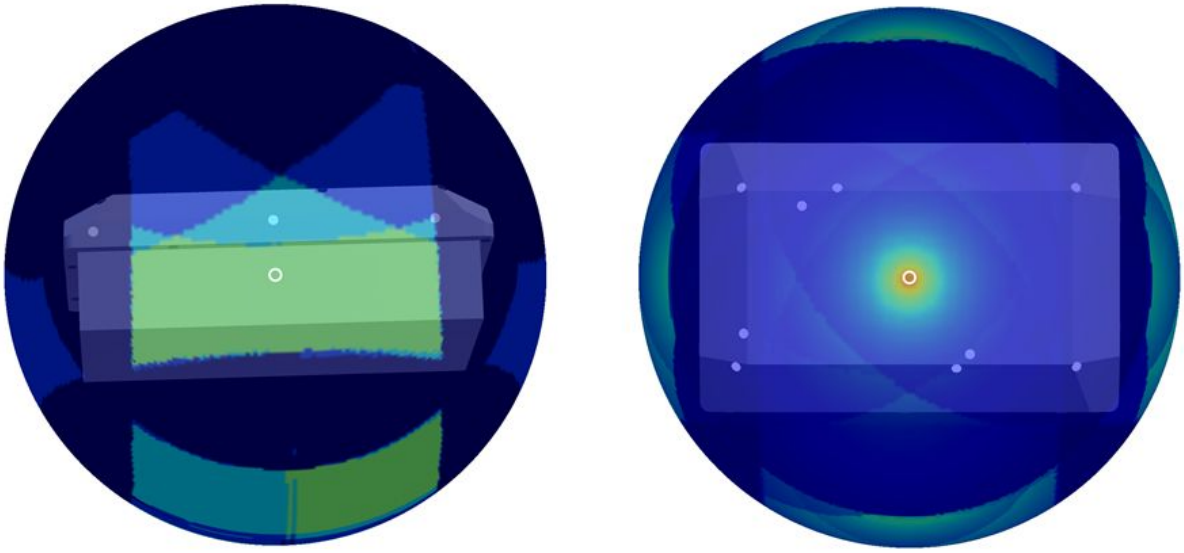
Interpreting Simulation Output

Inspecting the simulation output using the 2D and 3D plots generated by hmd_designer helps identify areas to improve sensor placement. Although the 2D and 3D plots show the same data, they are both useful in different ways. The 2D plots are very useful for quickly seeing the performance of an object and comparing the relative performance of different designs or design iterations. All simulated quantities and all orientations are visible at a glance. The 2D plots are also easily attached to emails and embedded in presentations, because they are rendered as PNG files.

The 2D plots represent simulation data for the Number of Visible Sensors, Initial Pose Possible?, Pose Rotation Error, and Pose Translation Error. These plots relate directly to the object’s ability to meet the three criteria for high performance tracking. When reading the plots, blue indicates good performance and brown means poor performance. Green represents marginal performance. The 2D plots are spheres that have been mapped onto a plane. It takes a moment to see how the three axes map onto the 2D plot.



The 3D plots are particularly useful for locating problem areas on the object. The simulation output forms a sphere around the object and allows the designer to orient the plot in such a way that the problem area appears directly over the object orientation causing the problem. The 3D viewer also makes it easy to see the effect of obstructions on tracking performance. Finally, the 3D viewer accounts for the 60° viewing angle of the photosensors, and highlights the sensors that are visible in any given pose.



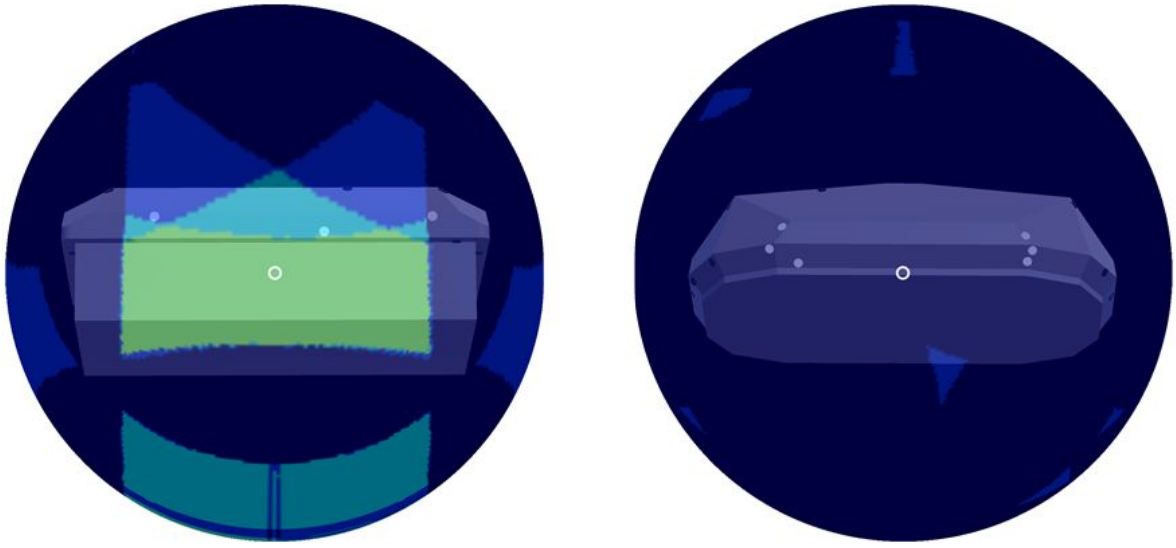
See the document **Interpreting Simulation Output** for more information. After identifying areas where tracking performance could be improved, it is time to refine the shape of the object to better support optimal sensor placement.

Refining the Shape

An iterative process of modifying the shape of a tracked object based on simulation results and resimulating to verify performance helps designers quickly refine object concepts and make tradeoffs. If the simulation output shows bad or marginal results at a particular orientation, the shape should be modified to improve sensor placement.

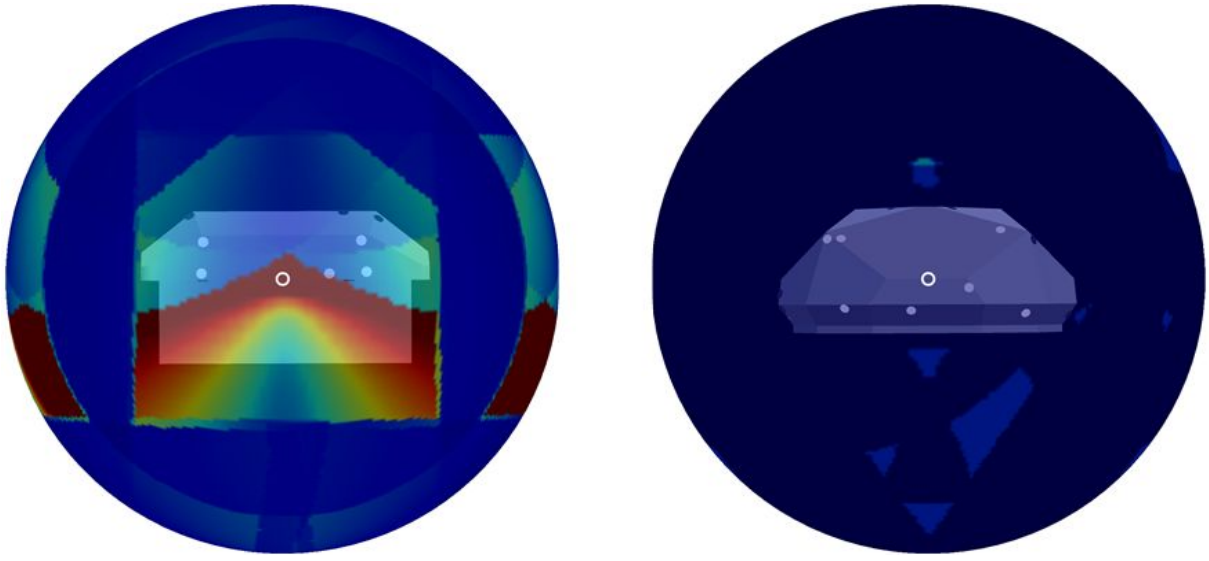
Visible Sensors

Achieving five visible sensors in any pose is a good first goal. Sometimes, that means opening up a face that was completely masked in the original design, to allow sensors to face in other direction. Other times, it may be necessary to add a facet to an edge, taking advantage of the ±60° viewing angle of the sensor.



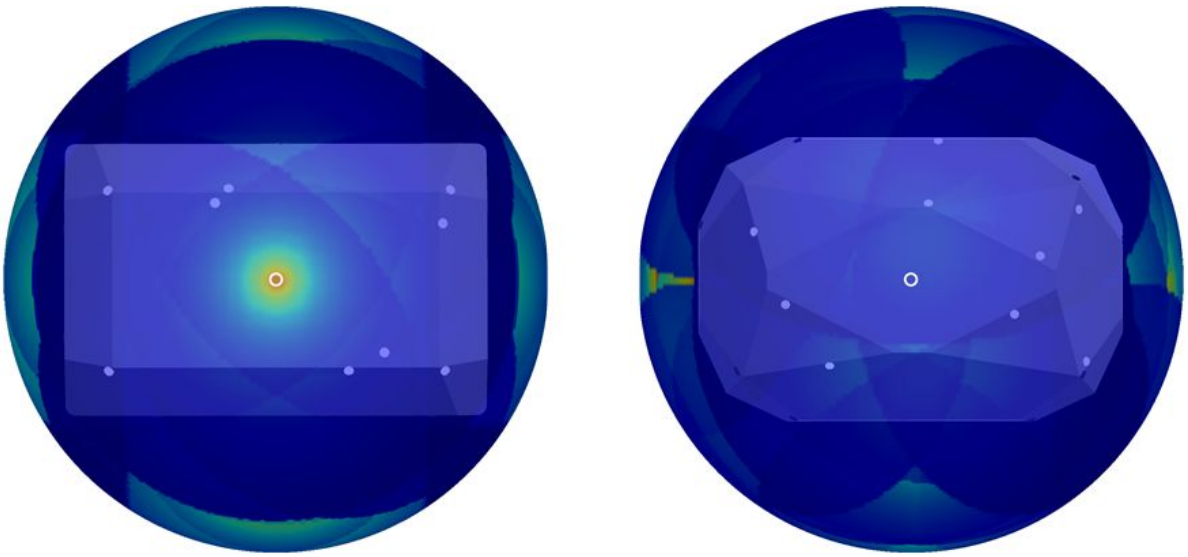
Translation error

If a design does not include enough baseline between sensors in some orientations, it may be necessary to increase the width of the object. Wider objects allow more baseline between sensors, and reduce translation error. Remember that translation error is driven by displacement away from the base station. Reducing translation error increases the object's operating distance from the base station.



Rotation error

Rotation error is driven by a lack of baseline in depth. Flat surfaces with no visible sensors beyond them are prime opportunities for rotation error. Increasing the baseline in all axes may be achieved by adding features to lift sensors out of a plane, adding features behind the plane, extending the depth of facets, or curving a surface to break the plane. If you are considering a curved surface, remember the rules for sensor covering described in the **Sensor Covering** document.



Tips and tricks for refining shapes to optimize performance are detailed in the document **Object Shape and Sensor Placement**. Once the basic shape of an object meets the tracking criteria, it may be possible to reduce the number of sensors, or arrange extra sensors to minimize occlusion. It is also time to take the positions suggested by the design software and place them in actual locations within the real mechanical design.

Defining Sensor Placement

Optimizing Sensor Placement

Although hmd_designer automatically generates locations for up to 32 sensors, it is highly unlikely that an engineer would use those locations directly. Actual sensor placement must consider industrial design, the overall mechanical and electrical design, and manufacturability. There are several steps to defining the final number of sensors and their locations on the tracked object. One of the first opportunities is reducing the total number of sensors.

Reducing the sensor count is possible if the output of the simulation shows redundant sensors. Especially with smaller objects, hmd_designer may place sensors very close to one another. Especially if these sensors are in the same plane, it may be possible to eliminate one of the sensors. It is easy to investigate the use of fewer sensors by reducing the count in hmd_designer and comparing simulation results. Reducing the number of sensors is one way to save cost. However, it is also an opportunity to add sensors back into the design to improve performance.

Integrating Sensors into the Mechanical Design

It is important to keep in mind that users will interact with their tracked objects in ways that may occlude sensors. Users may grip an object to adjust it, wave a hand in front of it, or use it in ways the designer never expected. Improving the robustness of the design may include adding extra sensors back into the design to

account for instances where some sensors are occluded. Of course, even with the optimum number of sensors chosen, their locations must be refined to accommodate the industrial and mechanical design of a manufacturable product. At this point, the designer begins using the output of hmd_designer as a guideline for integrating sensors into the mechanical design. This process is described in the document **Object Shape and Sensor Placement**.

Writing the JSON File

Realizing the industrial design intent, accommodating internal mechanical features, and achieving cost-effective electrical interconnect drive the final positions of the sensors in the tracked object. The sensor positions are no longer defined by hmd_designer, but input into the tool for visualization and simulation. Remember that one of the output files created by hmd_designer during the sensor placement generation process is a JSON file that describes the location and orientation of each sensor. Once sensor placement is defined by the engineer, the designer must create a JSON file. The JSON file holds three critical arrays pertaining to sensor placement:

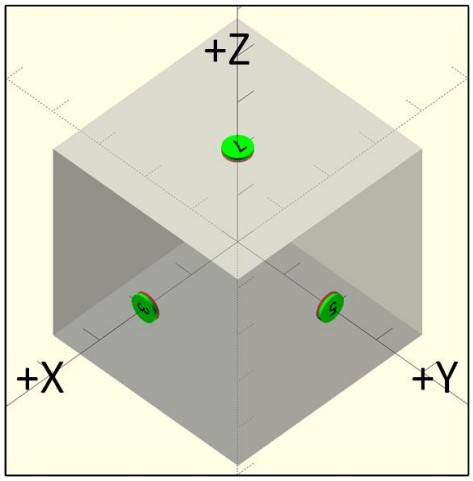
1. “channelMap” - indicates the port number on the FPGA where the sensor must be connected.
2. "modelNormals" - holds a unit vector normal to the surface of the sensor’s photodiode.
3. "modelPoints" - holds a position vector to the point in the center of the photodiode’s active area.

Remember that all position values in the JSON file are expressed in meters.

```
{
	"channelMap" : [3, 5, 7],
	"modelNormals" : [
		[1, 0, 0],
		[0, 1, 0],
		[0, 0, 1]
	],
	"modelPoints" : [
		[0.02, 0, 0],
		[0, 0.02, 0],
		[0, 0, 0.02]
	]
}
```

Visualizing the JSON File

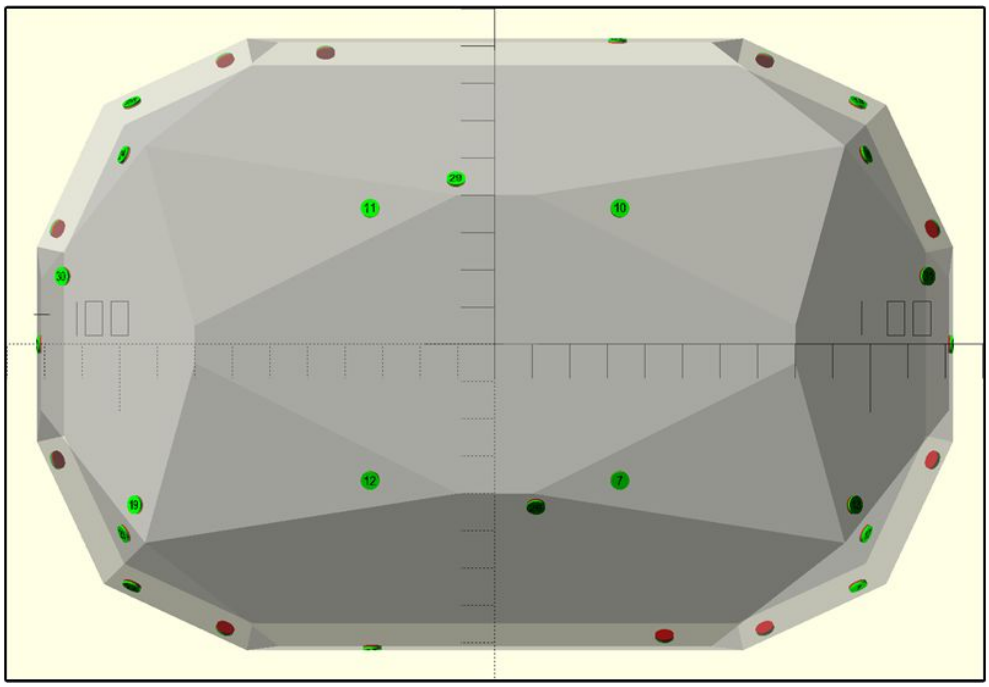
Working with the JSON file is a critical skill in tracked object design. Fortunately, the simulation tool also allows designers to verify their JSON files as they update the design. The simulation tool takes a JSON file and STL sensor object and renders the model in OpenSCAD. The visualization shows sensors placed in the location and orientation specified in the JSON file, and also annotates each sensor with the channel number.

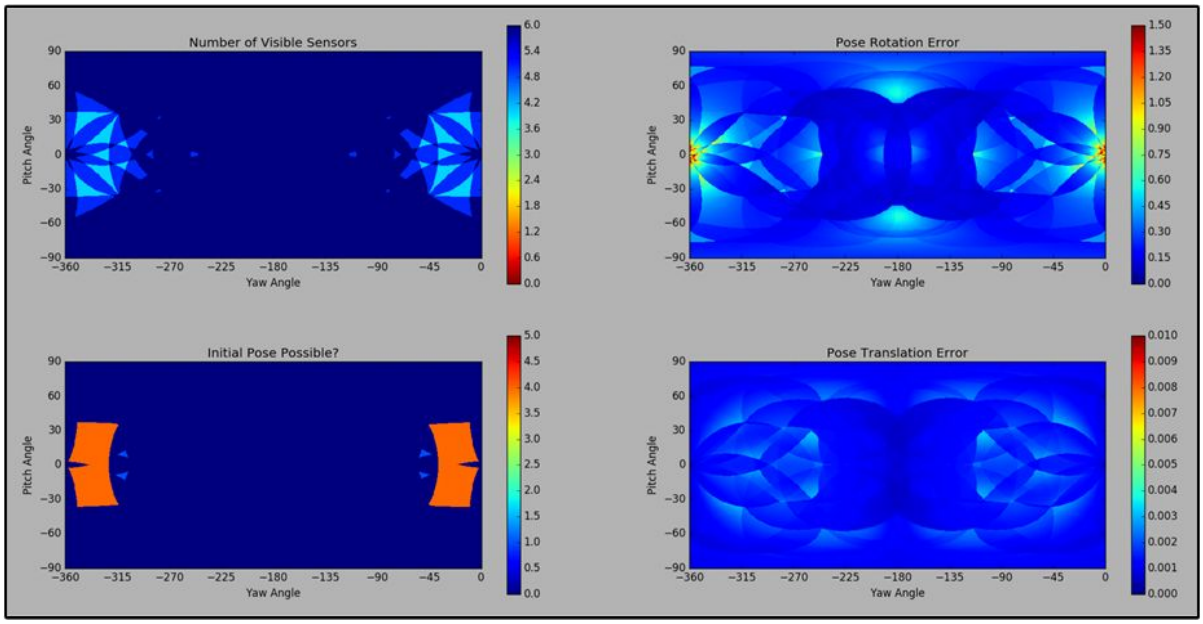


The JSON file's "imu" variable is described in the document **The JSON File**, and the process of visualizing JSON files is outlined under "Visualizing an Existing Model" in the **Simulation User Manual**. After the JSON file accurately reflects the design, it is important to verify the performance of the object has not suffered as the sensor locations were refined to their actual locations.

Simulating the Final Model

Verifying that a design still achieves acceptable tracking performance as the mechanical design matures is an iterative process. As the design matures the 3D CAD model changes, the JSON file is updated to reflect the new positions, and the design is simulated to verify tracking performance. If tracking problems are introduced, sensor locations may change again and the process repeats. Importing an STL file that defines the current shape of the object along with the associated JSON file with sensor positions into hmd_designer allows a user to simulate the design. Comparing simulation output with previous design iterations and objects with known performance keeps tracking quality up to par. The process of simulating an existing JSON file is described under "Simulating an Existing Model" in the **Simulation User Manual**. Of course, it is easy to imagine how tedious it could become to manually translate the 3D model into the JSON file for every design iteration.





JSON Automation

Although it is possible to measure the sensor locations and orientations in 3D modeling software and manually enter the values into the JSON file, the process is tedious. One valuable tool is a script that extracts the sensor positions and normals from the 3D CAD software. After automating the process for a given solid modeling package and workflow, the process of iterating on designs becomes much faster and less error prone. Tips and tricks for setting up solid models to facilitate measuring sensor positions and scripting the extraction of sensor data are documented in **Extracting Sensor Data from CAD**.

As the design finalizes, there is no better test of tracking than creating a prototype, integrating it into SteamVR™, and experiencing its performance first hand.

Rapid Prototyping

Rapid prototyping is the key to verifying the true tracking performance of an object. Effective evaluation requires a mechanical configuration that places sensors in the correct location, and an electrical system to interface the sensors to SteamVR™. The mechanical prototype is easiest to produce using 3D printing methods. Creating an initial solid model adapted to a 3D printing process is an important step, before refining the design to accommodate injection molding. Printing the 3D shape and mounting sensors either inside or on the surface, is acceptable to evaluate tracking. However, these sensors need an electrical connection to SteamVR™.

The SteamVR™ Tracking development kit endeavors to provide a designer with all the electronics required to create a rapid prototype without needing to fabricate printed circuit boards. Individual sensor PCBs may be applied to the 3D printed shape. Those sensors connect back to the evaluation board (EVM) via individual FPC jumpers, which are available in various lengths through retail distribution channels. The EVM connects to the SteamVR™ computer over USB or wireless.

Learn more about methods and recommendations for rapid prototyping in the document **Rapid Prototyping Objects**. Once the prototype is constructed, the designer can complete the JSON file and begin testing the prototype.

Completing the JSON File

Each tracked object contains a JSON file that describes its geometry and other properties required by SteamVR™. The simulation process requires the designer to specify the JSON variables that pertain to sensor placement, but there are more variables in the JSON file that are necessary for SteamVR™ integration. A complete description of the JSON file and how to specify the values it contains is documented in **The JSON File**. The highlights are mentioned below. Remember that all position values in the JSON file are expressed in meters.

Copy the sensor arrays into the “lighthouse_config” variable.

```
"lighthouse_config" : {
   "channelMap" : [3, 5, 7],
   "modelNormals" : [
      [1, 0, 0],
      [0, 1, 0],
      [0, 0, 1]
   ],
   "modelPoints" : [
      [0.02, 0, 0],
      [0, 0.02, 0],
      [0, 0, 0.02]
   ]
},
```

Set the IMU position and orientation using the “imu” variable using the “plus_x”, “plus_z” and “position” fields. Use hmd_deginer_gui to visualize the position and orientation of the IMU in the object. More information about visualizing JSON files may be found in **Simulation User Manual**.

```
"imu" : {
   "acc_bias" : [ 0, 0, 0 ],
   "acc_scale" : [ 1, 1, 1 ],
   "gyro_bias" : [ 0, 0, 0 ],
   "gyro_scale" : [ 1, 1, 1 ],
   "plus_x" : [ 1, 0, 0 ],
   "plus_z" : [ 0, 0, -1 ],
   "position" : [ 0.002, 0.010, -0.010 ]
},
```

It is easier to troubleshoot initial tracking performance by viewing any tracked object as a controller. Set the “device_class” to controller and set the “render_model” to a working render model.

```
"device_class" : "controller",
"render_model" : "generic_hmd",
```

The “head” member defines the relationship between the tracked object’s coordinate system and the SteamVR™ coordinate system. It is used differently for HMDs and controllers. See **The JSON File** for more details.

```
"head" : {
   "plus_x" : [ -1, 0, 0 ],
   "plus_z" : [ 0, 0, 1 ],
   "position" : [ 0, 0, 0 ]
},
```

After initial tracking is verified as a controller, an HMD may be evaluated by changing the “device_class” to “hmd.” To function as an HMD, the “head” variable must reference the coordinate system of the model to the point between the wearer’s eyes. There are also several HMD-specific parts of the JSON file related to the display and optics, which must be completed as described in **The JSON File**.

The values described above are necessary for testing the object’s tracking performance. However, before an object is tested it is important to complete all of the entries in the JSON file. Refer to **The JSON File** for a complete description of the file and its contents.

After finishing the JSON file, use the lighthouse_console utility to upload the file to the tracked object.

> **Tip:** It is strongly recommended that you disconnect all other SteamVR™ devices when using uploadconfig. If you overwrite the configuration of other objects they can be rendered inoperable. Consider using downloadconfig on all devices and archiving the JSON files to enable recovery.

```
lh> uploadconfig c:\<folderlocation>\<filename>.json
```

When the object is built and the JSON file has been uploaded, it is ready to connect to SteamVR™ over USB. It often saves time to run a few simple tests in lighthouse_console before trying to work with the object in VR. Also, it is important to calibrate the object before evaluating the tracking performance.

Testing Object Connectivity

The lighthouse_console utility provides some simple but effective commands you can use to verify optical sensors, the IMU, and communication paths are functioning properly.

- Test USB connectivity by connecting the the object using lighthouse_console.
- Use the “dump” command to output sensor information using the “sample” command. Look for streaming sensor hits from all the connected sensors.
- Use “dump” with the “imu” command and verify that IMU data is streaming appropriate values at 1000 Hz.
- Use the “period” command to verify sensor hits from all the object’s sensors.

Refer to the document **Testing Object Connectivity** for a full description of how to run the tests and interpret the results. Once the sensors and IMU are streaming data over USB, the device is ready to integrate into SteamVR™. However, it is important to calibrate the object before verifying the tracking performance.

> **Tip:** An object must boot before it may be optically calibrated. Connect the object to SteamVR™ and debug the object and JSON file until the device tracks in VR. The upcoming Optical and IMU calibration improve tracking performance, but are not required for initial tracking.

Optical Calibration

Optical calibration is a process of measuring the actual positions of sensors on the tracked object and updating the JSON file to reflect the exact sensor placement. Manufacturing tolerances guarantee that the sensor placement on a given device never exactly match the ideal values specified in the JSON file. Valve has written a tool that takes the ideal JSON as a starting point and uses the reference signals from a base station to measure the exact locations of sensors on an object. The tool produces a calibrated JSON file specific to the calibrated device. It is important to perform this calibration process for each new device, whether in the lab or in mass production.

With a base station active, the tool vrtrackingcalib uses reference signals from a base station to measure the actual sensor locations on a device. Remember these rules for proper calibration:

1. Always start from the ideal JSON file. Do not calibrate from a JSON that was generated by vrtrackingcalib, or errors could accumulate.
2. The object must have a JSON uploaded before it can be calibrated. That JSON can be the ideal JSON, or the JSON from a previous calibration step.
3. Ensure that only one base station is active during calibration. Calibration will proceed faster if this base station is in mode “A”.
4. Move around in the volume to average out basestation errors.
5. Use different object orientations to average out other system errors.
6. Show many different overlapping combinations of sensors to the base station.

```
Realigning point cloud to original model:
> RMS: 0.000266
> Ceres Solver Report: Iterations: 14, Initial cost: 1.885440e-007, Final
cost: 1.767502e-007, Termination: CONVERGENCE

> Corrected scale of 0.9999
> Corrected normal shift of 0.0001
> Corrected rotation of 0.0977 deg
> Corrected translation of 0.0636 mm
```

After vrtrackingcalib produces a new JSON file with updated positions in the “modelPoints” variable, upload the new JSON file to the tracked object using lighthouse_console. All other values in the JSON are preserved from the JSON that was previously uploaded to the device. For documentation of the full optical calibration procedure refer to **Optical Calibration**.

IMU Calibration

IMUs exhibit offset and bias errors as a result of the fabrication process. SteamVR™ is capable of calibrating these errors out of the system. However, it can take time after tracking starts to reach the correct value. Valve has written a utility, imu_calibrator, that measures the IMU errors in a device. The calibration tool produces a snippet of JSON that may be added to the “imu” variable in the JSON file. Starting from the measured offset and bias reduces the time it takes SteamVR™ to converge on the optimal IMU error values.

The calibration process involves placing the object in six orthogonal positions, while the calibration utility takes IMU measurements. When calibration completes, the snippet of JSON is ready to cut and paste into the object’s JSON file.

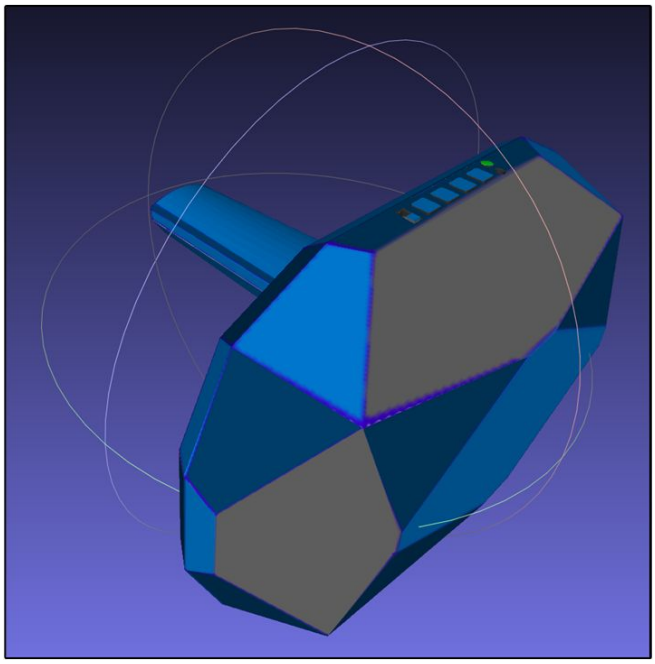
```
"acc_scale" : [ 0.998, 0.9983, 0.9915 ],
```

```
"acc_bias" : [ 0.05089, -0.03676, -0.2253 ],
"gyro_scale" : [ 1.0, 1.0, 1.0 ],
"gyro_bias" : [ 0.06253, 0.01054, -0.02128 ],
```

The tested and calibrated object is now ready for a deeper evaluation of tracking performance, and refining the object's design. However, to see the object in SteamVR™, it requires a render model. At first, the render model could be one that already ships with SteamVR™; but, having a render model that matches the shape of the device helps designers interact with the object in VR.

Creating a Render Model

An object's render model is the 3D image that appears in VR to represent the object. The default render model is referenced in the objects JSON file, and typically appears as the tracked object appears in the real world. Having a render model that matches the physical object makes it easy for the user to pick up the object when viewing it in VR. Once the user opens an application in SteamVR™, the application may override the default render model to make the object appear as a hand, weapon, tool, creature, or any other entity the experience requires.



The render model consists of a minimum of three files:

1. Object file (.obj)
2. Material file (.mtl)
3. Texture file (.tga)

These files are generated by 3D artists using specialized 3D graphics software packages. SteamVR™ Tracking documentation is not intended to provide a complete course in any 3D graphics software package. However, it does provide one basic example using an open source software package called Blender™. The process of creating a simple render model using Blender™ is outlined in **The Render Model**.

Once the files are generated, integrating them into SteamVR™ is as simple as placing them in a subfolder of the "rendermodels" folder and referencing the new folder in the object's JSON file. The specifics of that process are

documented in **The Render Model**. With the render model in place and the device connected over USB, it is time to start SteamVR™ and interact with the new object in virtual reality.

Evaluating Tracking Performance

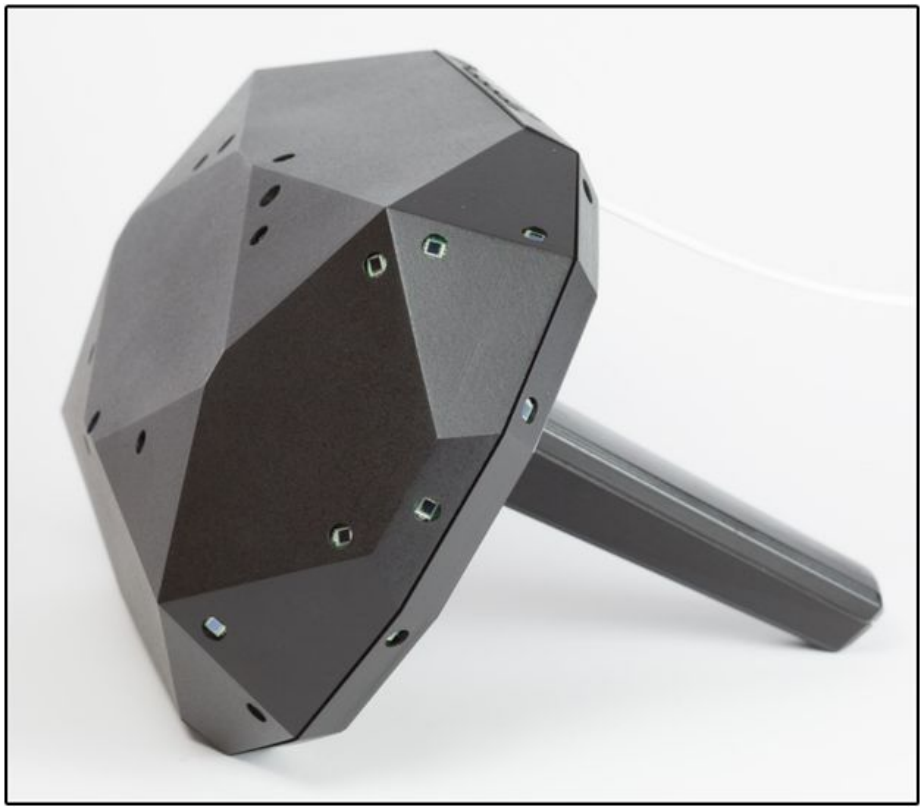
Effectively evaluating tracking performance and troubleshooting tracking problems requires experience. There is no way to quantify the performance of a tracked object. As the limits of the system expand and new objects are developed for different applications, the only way to truly verify performance is to experience it in VR. Valve highly recommends purchasing the latest SteamVR™ devices and spending time in VR to internalize how the highest performing devices feel. When a designer's senses are calibrated to good performance, it becomes far easier to notice slight errors in new devices. A complete guide to evaluating tracking performance is documented in **Evaluating Tracking Performance**. Generally speaking, evaluating tracking falls into three major categories:

1. Object booting - Does the object appear in VR and stay visible?
2. Controller tracking - Does the object track as a controller, with no visible vibration or distortion?
3. HMD tracking - Does the object track with low enough vibration and distortion to work as a head mounted display, without any visible or nausea-inducing effects?

It is useful to start the evaluation process with the device configured as a controller. Inspecting the object in VR through a known good HMD is a great way to see the tracking performance of the new object. Questions 1 and 2 above can be answered with the object configured as a controller. For objects attached to binocular displays, after any visible tracking problems are eliminated in controller mode, the object may be configured as an HMD and further investigated.

Using a rendermodel that matches the device under test makes it much easier to tell if it is operating properly. For this evaluation, you can use a rendermodel generated from CAD with a simple texture like a baked ambient occlusion map. A proper render model should be created from scratch using best practices for 3D graphics mesh design before shipping.

Evaluating an object as an HMD helps dial in the "head" variable in the JSON file, and also allows users to begin evaluating optics distortion and display characteristics in VR. Interacting optics, tracking, and display systems can make tracking problems more challenging to debug. However, the document **Debugging Tracking Issues** tries to chronicle known issues and resolutions.



Summary

The process of designing and integrating a tracked object into SteamVR™ is a multidisciplinary effort that spans initial conception through sensor placement, simulation, prototyping, calibration, software integration, evaluation and troubleshooting. Always keep in mind the criteria required for high performance tracking, and impress upon industrial and mechanical designers the impact of object shape, materials, and finish on tracking quality. Hmd_designer is a powerful tool for generating sensor placement and simulating results while refining the object design. Once the shape of the tracked object is mature, placing sensors in the 3D CAD model, exporting those positions and orientations to the JSON file, and simulating the revised design to verify performance is an important iterative process. It can be very useful to develop a script for the mechanical engineer's solid modeling software that extracts sensor positions and orientations to produce a JSON file. Simulating the design is important, but there is no substitute for rapid prototyping an object to test the actual shape. Integrating the prototype into SteamVR™ requires a completed JSON file to describe the object. JSON files are uploaded to the object using lighthouse_console. Lighhouse_console is also a useful tool for testing basic connectivity before moving on to SteamVR™. Optical calibration, using vrtrackingcalib, optimizes tracking performance before testing. To see the object in VR, a render model is required. Engineers can start with one of the render models provided with SteamVR™. However, developing a custom render model that reflects the shape of the tracked object is very useful for testing. Finally, evaluating tracking requires experience. Being intimately familiar with the performance of existing SteamVR™ products is the best way to develop the experience required to evaluate the tracking performance of a new device. Troubleshooting tracking problems is challenging, but SteamVR™ logs and documentation can help identify problems so that new tracked objects reach their full potential.