User Interface Guidelines

Preview Version 1.0

zSpace, Inc.
490 De Guigne Drive, Suite 200
Sunnyvale, CA 94085
(408) 498-4050

Contents

Preface

Who Should Read This?

The zSpace User Interface Guidelines are for visual and interaction designers, as well as developers, product managers, and program managers who are responsible for designing an application for the zSpace system. This can be a new application or a port of an existing application.

Topics in These Guidelines

These Guidelines include the following chapters:

"Introduction to the zSpace Platform" on page 7: This chapter introduces you to 3D basics, describes stereoscopic 3D in the zSpace system, and suggests the best uses for the zSpace system.

"Fundamental Design Principles" on page 13: This chapter provides the key design principles that define good application interfaces.

"Design Guidelines" on page 19: This chapter provides guidelines on application-level UI layout, content layout, and user input.

"Design Considerations" on page 35: This chapter covers topics that are very application-specific, such as whether or not to include real world physics and how to use special effects. Other topics include navigation, performance, responsiveness, and universal design.

"Stereo Optimization" on page 45: This chapter provides information on how to optimize stereoscopic 3D by adjusting settings in the zSpace SDK.

Terminology and Conventions

These Guidelines use **bold** for new terms and *italics* for emphasis. New terms will also appear in the Glossary, so you can look them up later.

The term **3D** is often applied to drawings or applications that give the illusion of 3D with perspective and shading. In these Guidelines, unless otherwise specified, 3D specifically refers to **stereoscopic 3D**, as it is perceived in the real world and in the zSpace display.

These Guidelines use the term **viewpoint** to refer to what the user can see on the display. Another term for this is the primary camera.

The **zSpace UI Toolkit** is a Unity-specific SDK that will help you implement many of these design guidelines. This is currently under development.

The **zSpace SDK's Core libraries** include a set of parameters and functions that support the 3D display, head tracking, and the stylus. These functions are available both in C++ APIs and Unity-specific APIs.

Source Material

These Guidelines are based on existing 2D design guidelines, existing 3D guidelines, and our own experience with zSpace applications. Specific resources include:

- OS X Human Interface Guidelines
- Designing with the Mind in Mind by Jeff Johnson
- 3D User Interfaces: Theory and Practice by Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola, Jr., and Ivan Poupyrev

Chapter 1 Introduction to the zSpace Platform

This chapter covers the following topics:

Understanding 3D Basics

This section provides an overview of how you experience 3D in the real world. This starts with binocular and stereoscopic vision, with additional depth cues from motion parallax and proprioception. If this sounds familiar, you can skip ahead to the next section.

You see things in three dimensions because you see slightly different views from each eye. The ability to see out of both eyes simultaneously is called **binocular vision**. Assuming the objects are not too close, your brain fuses the two views into a single view. **Stereoscopic vision** is the ability to perceive that single image in three dimensions.

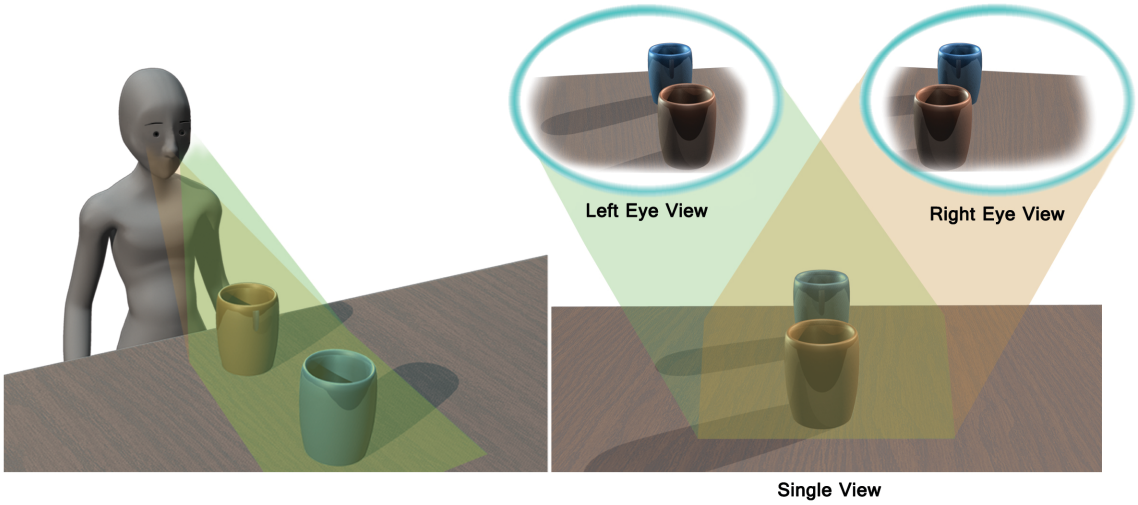

*Figure 1-1 **Binocular Vision***

Another key aspect of stereoscopic vision is that the view changes when you move your head. As an example, line up two objects on your desk, such as two coffee cups, one behind the other. Move your head to the left and right and notice that the back coffee cup appears and disappears as your viewpoint changes. The difference in perspective from one head position to the next is called **motion parallax**. Another aspect of motion parallax is that objects in the distance appear to move less than objects closer to you. Without motion parallax, you can experience cognitive dissonance, leading to fatigue and discomfort.

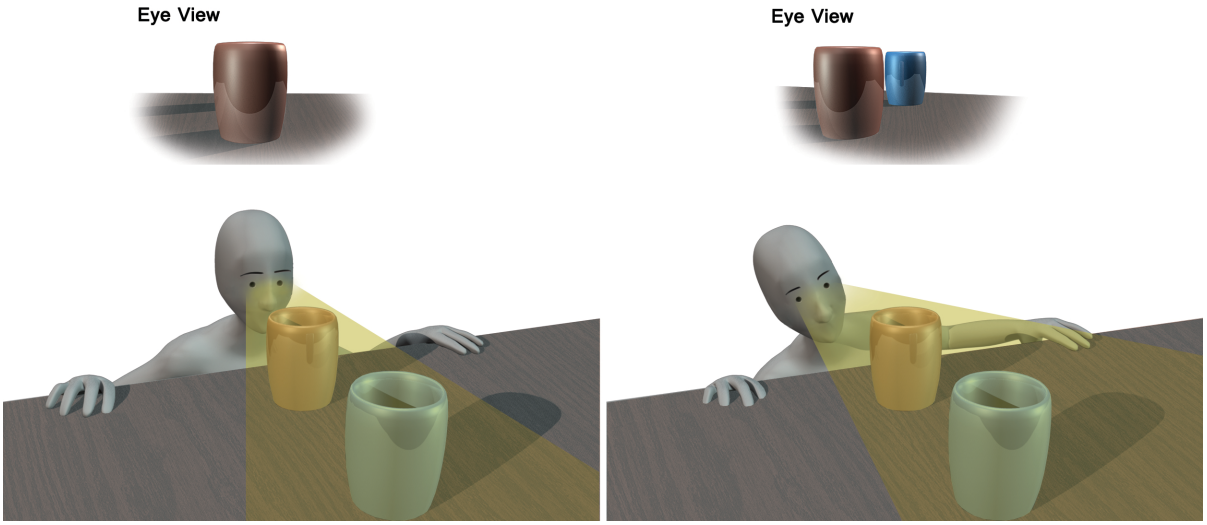

*Figure 1-2 **Motion Parallax***

We also receive cues about the 3D world through our physical interaction with it. Knowing your body's position is called **proprioception**. Reach out to touch your coffee cups. You reach further for the second coffee cup. This action reinforces both your depth perception and your understanding of the two objects' positions relative to each other.

Experiencing 3D in zSpace

This section describes how you experience 3D in the zSpace system. Note that the zSpace display shows true stereoscopic 3D. This is different from – and better than – other types of 3D representations, such as 3D movies and 2D or monoscopic displays of 3D objects. Even if you have experienced the zSpace system, you should skim this section for new concepts, in bold.

Note: Throughout these Guidelines, we use both the term stereoscopic 3D and 3D to refer to the zSpace display of 3D objects. When we discuss traditional 3D in a 2D environment, we will refer to 2D or monoscopic displays.

Stereoscopic Vision in zSpace

You experience stereoscopic vision in the zSpace display much like in the real world. The zSpace system displays offset images for the left and right eye. The zSpace system includes passive polarized glasses to ensure that each eye only sees a single image. Your brain fuses the images together, producing a single stereoscopic 3D image.

Objects in the zSpace display can appear at different depths. When something appears to be at the exact depth of the screen, or on the screen's surface, this is **zero parallax**. Objects that appear in front of the screen, projecting towards the viewer, are in **negative parallax**, while objects that appear behind or inside the screen are in **positive parallax**.

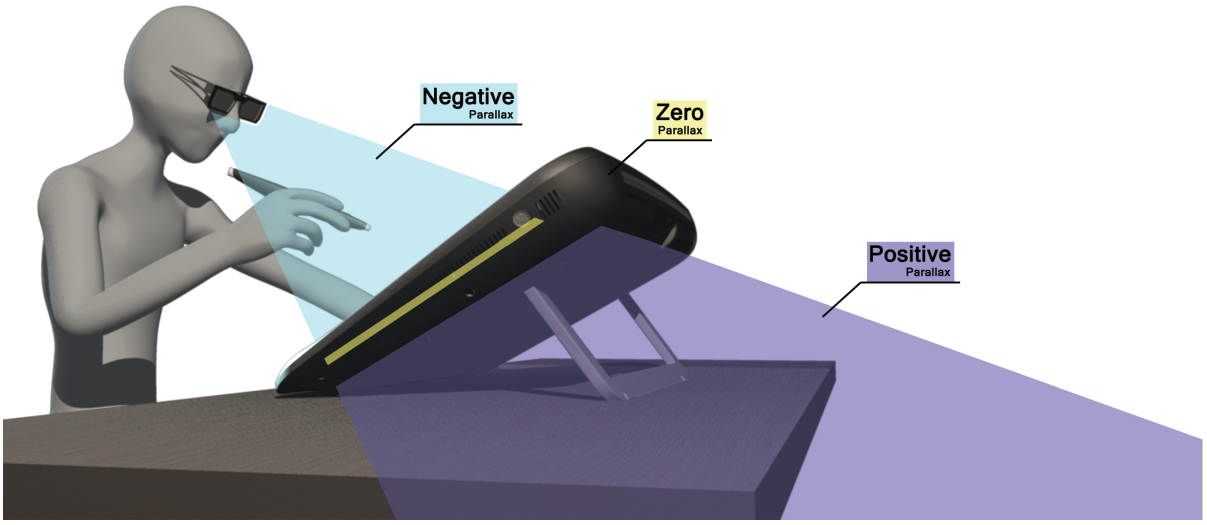

*Figure 1-3 **Negative, Zero, and Positive Parallax***

If you have seen a 3D movie, you are familiar with this. Some things in the movie project towards you (negative parallax) and some appear to be further away than the movie screen (positive parallax). However, depending on where you were sitting, you might have noticed some distortion.

Unlike 3D movies, viewing the zSpace display is not subject to distortion based on where you are sitting because it provides motion parallax.

Motion Parallax in zSpace

As discussed in "Understanding 3D Basics" on page 7, motion parallax is a key aspect of how we perceive 3D in the real world. Sensors on the polarized glasses allow the zSpace system to track the position of your head. When you move your head, the zSpace system adjusts the viewpoint accordingly. Although the view changes, the objects still appear stationary. This provides motion parallax, which is critical because your brain expects the view to change. If it does not, this can lead to visual or mental fatigue.

In addition to providing a stereoscopic 3D display and motion parallax cues, the zSpace system includes proprioception cues.

Proprioception in zSpace

The zSpace system lets you directly interact with the 3D scene via a stylus. Knowing your body's position (proprioception), combined with manipulating an object in the scene, reinforces your sense of depth. The additional depth cues make the zSpace display easier to view and comprehend than a 3D movie or a 2D rendering of 3D objects.

The stylus also offers some distinct advantages over a mouse:

- The stylus provides six degrees of freedom, while the mouse is more limited. In other words, the stylus can provide the application with x, y, and z coordinates about its location, as well as its angle (yaw, pitch, and roll).

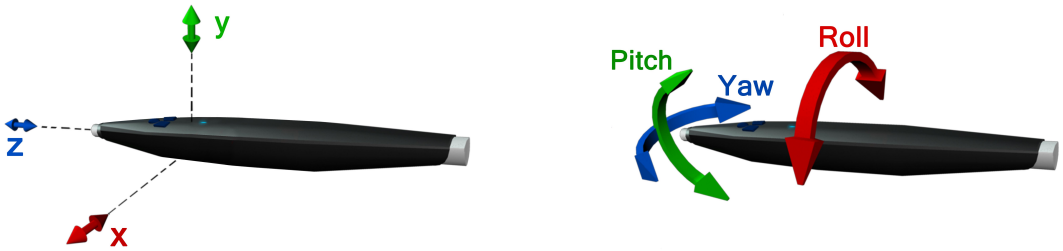

*Figure 1-4 **Stylus Movement***

A standard mouse can only provide the application with x and y coordinates on a single plane (your desk or mouse pad). Note that this limitation does not apply to 3D mice such as SpaceNavigator.

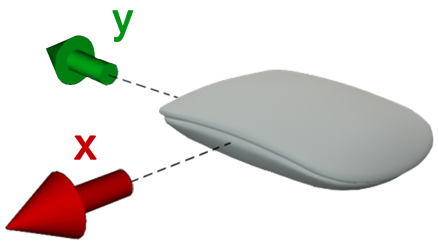

*Figure 1-5 **Mouse Movement***

- The stylus provides a natural interface for selecting objects at different depths. While it is possible to use the mouse in stereoscopic 3D, it is not designed for this purpose.
- The stylus provides a more intuitive interface for interacting with 3D objects than the mouse does.

Making the Most of zSpace

Some cognitive psychologists believe that interactive, stereoscopic 3D systems, such as the zSpace system, use the intuitive reasoning system, which is faster than our analytic reasoning system[1]. Thus, in certain applications, we can more quickly perceive and understand data when it is displayed in 3D. This applies to both real world objects such as parts in a discrete manufacturing process, as well as representations of abstract data.

Based on our experiences in the real world, we also have a natural ability to interact with objects in stereoscopic 3D. Using the stylus to pick up objects, bring them closer to us, and rotate them to examine them is an easy extension of using our hands in the same way.

You can port or create any type of application into the zSpace system and benefit from the ability to display and manipulate objects in three dimensions, complete with spatial relationships to each other and to the viewer. Some applications depend more heavily on users' spatial reasoning. Spatial reasoning involves evaluating the visual details of objects and their relative positions. Example applications include:

- Evaluating designs for manufacturability and consumer appeal.
- Learning physical layouts, configurations, and functions, possibly in mechanical design or physiology.
- Detecting physical anomalies, such as in disease diagnosis.

---

[1]Patterson, R. & Silzars, A. (2009). Immersive stereo displays, intuitive reasoning, and cognitive engineering. *Journal of the Society for Information Display, 17*, 443-448.

The zSpace system offers a significant advantage whenever it is either too expensive or not feasible to interact with the physical object. Examples include:

- Rendering designs of parts and products in stereoscopic 3D before they are physical prototypes can lead to a significant cost savings.
- In the medical and biotechnology areas, modeling organs or molecules in stereoscopic 3D is very valuable.
- For large amounts of data, such as produced by scientific experiments or simulations, stereoscopic 3D graphics can make it easier to interpret the data and discover patterns.

For More Information

For more information on 3D perception and the implications for the zSpace system, refer to one or both of the following:

- Interactive Stereoscopic Displays white paper
- How zSpace Dramatically Improves Creativity and Understanding webinar

Chapter 2 Fundamental Design Principles

Before we begin with the fundamental principles for designing in the zSpace system, we have to answer this question: If the zSpace system excels at stereoscopic 3D, should everything be rendered in 3D? The answer is no, not everything.

We believe some things are better displayed in 2D, based both on complexity and the users' past experiences with 2D interfaces. For example, it is far easier to read text in 2D than in stereoscopic 3D. Users will be more efficient with user interface controls in 2D because the physical interaction is less complex in two dimensions.

However, you will be designing your application's scenes and objects in stereoscopic 3D. In many cases, it is far more powerful to model the real world and leverage users' real world experience in 3D. This taps into our ability to understand certain things better in 3D. For example, we can more readily experience and understand spatial relationships and physical constraints in 3D.

Because your application will include both 2D and 3D, we base our fundamental principles on users' past experience with 2D user interfaces and past experience with the 3D world. Your application will be more successful when it builds on user experience and knowledge.

As you read the following principles, remember that a stereoscopic 3D environment is both richer and more complex than a 2D environment. Your design should be only as complex as necessary for the user to accomplish the task. Keep UI elements separate from your scene as much as possible. Do not add unnecessary features. For example, although a 3D environment can support rotating, resizing, and repositioning an object, not every task requires all three options.

Metaphors and Direct Representation

Traditional 2D user interfaces employ a number of familiar metaphors, such as the file folder to represent a directory. To leverage a user's 2D UI experiences, follow the existing conventions. In general:

- Do not reuse an existing metaphor or object in a new way.
- Do not invent a new metaphor or representation for one that already exists.

To leverage a user's experience in the real world, you can use a direct representation of an object instead of a 2D metaphor. Use 3D objects instead of 2D metaphors when it makes sense *in the context of your application*. For example, in a photography application, you could set the controls for lighting via dialog boxes. However, if your scene depicts a photography studio, you could also set lighting objects in the scene. The user would adjust the lighting by moving the light objects.

Note: Make sure that replacing a metaphor with an object enhances your application, rather than over-complicates it. For most applications, creating a 3D file folder object for directory navigation would add unnecessary complexity. Users are already familiar and comfortable with the 2D file folder metaphor.

Mental Model

Your application should reflect the user's experience with the applicable tasks and objects in the real world. In some cases, stereoscopic 3D allows for a very realistic display of objects in the real world, such as a human heart. In other cases, such as data visualization applications, you will be using a more abstract representation. In all cases, consider these points:

- **Familiarity:** Use symbols and terminology that the user already knows. Take advantage of any standardized 2D icons and menu groups, such as File -> Save.
- **Simplicity:** Focus on the basic tasks that need to be accomplished. For the user's task, choose key components from the real world. Do not include extraneous objects.
- **Availability:** Make key features readily available. While you want an uncluttered user interface, do not make advanced features difficult to find.
- **Discoverability:** Make it easy for the user to identify objects and UI controls available for interaction. For example, some objects in your scene may be static, while the user can reposition and resize others. Provide cues that help the user see the difference. You can leverage the user's experience with 2D, in which unavailable menu items and toolbar options appear in gray.

Direct Manipulation

Direct manipulation refers to user interactions with an object. Based on past experiences, the user already knows how to directly manipulate objects in both 2D user interfaces and the real world.

The zSpace system offers a rich platform for users to directly manipulate objects in stereoscopic 3D. For example, users can rotate, reposition, resize, or zoom on an object. In the zSpace system, user interactions can be more intuitive because we learn from birth how to interact with the 3D world.

Take advantage of the stylus to directly manipulate objects in stereoscopic 3D whenever you can. Remember that the stylus supports a wider range of movement than the mouse. Refer to "Proprioception in zSpace" on page 10 for more details.

You can use constraints to make it easier for the user to complete specific tasks. In some cases, you may constrain input to fewer degrees of movement, such as limiting input to the x and y axes. Another example would be a snap-to-grid constraint. You can also give the user control over whether constraints are on or off. Note that snapping to a grid is a concept borrowed from 2D UIs. It will make sense to users with experience in a drawing application.

Your application can also take advantage of a user's experiences with common 2D actions such as dragging and scrolling. This is especially true for the user's interaction with the UI controls. If your application presents abstract concepts not found in the real world, you can leverage the user's experience with these concepts, or similar abstractions, in 2D applications.

See and Point

See and point simply means that users should be able to see the objects and UI controls that they need, and they should be able to point to them. Where it is practical, you can build on the user's experience with these common 2D UI paradigms:

- Noun-then-verb: Select an object (the noun) and then the action to be performed (the verb).
- Dragging: Drag an object (a noun) to another object that performs an action (verb), such as dragging a file to the trashcan.

In the real world, users are very familiar with pointing at objects, as well as "grabbing" them. The zSpace stylus is a natural extension of users' experiences at pointing at objects using a pen, pencil, or laser pointer. For "grabbing" actions, users quickly build on their experiences using a mouse in 2D environments.

Whether your application takes advantage of 2D or real world models, make it clear to the user how to interact with the objects in the scene and how they will respond to the stylus. For example, 3D applications easily support scaling, rotating, and repositioning objects. You can use context menus, tooltips, and different manipulators to indicate the available actions.

User Control

In general, your application should give users control, rather than taking action for them. However, there are a few exceptions:

- Users expect certain automated actions based on their experiences with 2D user interfaces. A common example is autosave. Note that this is often configurable, so the user still has control.
- Users expect applications to prevent destructive actions. For example, if the user tries to close an unsaved file, almost all applications prompt the user to save first. The user regains control after acknowledging the warning.

When considering automated actions in stereoscopic 3D, you should evaluate whether the user might expect such actions based on either 2D UI knowledge or real world knowledge.

Feedback and Communication

Feedback can contribute to a user's sense that an application is fast (or responsive), even when the performance is slower. Always provide immediate feedback when the user takes an action. Build on the user's experience with 2D user interfaces by indicating which controls are selected or pressed. Similarly, show that an

object in the scene is selected by changing its look. For example, you can change the outline or display a wireframe.

You can also use haptic feedback provided by the stylus. When the stylus hovers over a control or in-scene object, a subtle vibration of the stylus helps the user correctly position the stylus for selection. For more information, refer to "Stylus Feedback" on page 42.

As the user acts on scene objects that represent the real world, you can provide feedback that matches real world behaviors. Depending on your application's requirements, you might model the behavior of objects based on material composition. For example, a rubber ball bounces when dropped, but a slice of toast does not. For more information, see "Real World Physics" on page 38.

When an action takes more than a second to complete, show either a busy cursor or a progress bar. Similarly, display clear messages when a task cannot be completed or an error occurs. This will match the user's expectations based on experience with 2D user interfaces. For more information on this topic, refer to "Improving Responsiveness" on page 41.

Make sure your application maintains spatial and temporal compliance between multiple feedback dimensions. In other words, visual, auditory, and haptic feedback should be consistent. For example, your application might use both visual cues and stylus vibration to indicate an object is selected. If the cues are not synchronized, this can cause confusion.

Consistency

Users will have more success using your application if it is consistent. Consider the following questions:

- Is your application consistent with the zSpace User Interface Guidelines?
- Is your application consistent with the user's mental models and known terminology, both in 2D user interfaces and in the real world?
- Is your application internally consistent? Does it use the same terminology and user interface throughout?

If possible, use the zSpace UI Toolkit[1] to ensure that your application behaves consistently with other zSpace applications.

WYSIWYG and Realism

The basic principle of WYSIWYG is that displays should match the final output. For example, display text as it will appear when printed or displayed on a web page. Display video as it will appear in its final form. Update the display to reflect changes immediately.

[1]The UI Toolkit is currently in development. The first platform will be Unity.

In stereoscopic 3D, if possible, use a 1:1 mapping scale, where objects are shown in their actual size. This has much more impact than showing an object at a different scale. For example, a full-size shoe in 3D *is* a shoe. However, a 3D car in reduced scale is a *model* of a car. Note that a stereoscopic 3D model still has greater impact than a 2D image. Where practical, aim for realism.

Make sure you show objects in the correct detail. In the real world, objects that are closer show greater detail than objects that are farther away. This applies to both your scene and the objects in it.

> **Note:** While realism in your scene is generally desirable, be guided by the task. Make objects photorealistic when they enhance the user's ability to accomplish the task. Do not apply realism to all possible objects for its own sake. For example, we recommend a 2D UI for application controls.

For more information on realism, refer to the following sections:

- "Content Layout" on page 29
- "Real World Physics" on page 38
- "Performance and Responsiveness" on page 40

Forgiveness

Make sure your application encourages exploration by offering mechanisms to undo actions, return to prior saved settings, and return to default settings. This is consistent with users' expectations based on 2D applications. For example, many users have memorized the keyboard shortcut for Undo.

Forgiveness is particularly important in a stereoscopic 3D application that lets users rotate, reposition, and resize objects. If a user does not remember the current mode, it is easy to resize an object when the user meant to reposition it.

Another part of forgiveness is preventing serious mistakes with warning messages. For example, if a user is about to exit without saving changes, display a message confirming the user's intention.

Perceived Stability

You can give users a sense of stability by providing a user interface that meets the users' expectations. You can achieve this by:

- Providing a consistent set of menu items and UI controls. If a menu item is not applicable at a particular time, change it to gray rather than hiding it completely. You can use context menus as well. This conforms to users' expectations based on 2D UI experiences.
- Saving the user's application settings from one session to the next. The user will be more comfortable and more efficient if the application looks the same each time it opens.

- Implementing real world behaviors for objects in your scene. Refer to "Real World Physics" on page 38 for more on this topic.
- Using the zSpace UI Toolkit, so that your application, in general, has a similar look and feel as other zSpace applications.
- Following these UI Design Guidelines.

Both the UI Toolkit and these Guidelines are based on zSpace's experiences building demo applications and supporting third-party developers.

Aesthetic Integrity

Aesthetic integrity refers to following good design principles. Note that good design principles are based on human cognition and apply equally to 2D and stereoscopic 3D interfaces. If you are new to UI design, refer to "Source Material" on page 6 for references on good design.

For 2D, use UI elements according to accepted standards. For example, use radio buttons for mutually exclusive options, and use checkboxes when multiple choices are possible.

Keep your user interface uncluttered. Although you could use 50 radio buttons to capture a state (such as California), a drop-down list box is much simpler. To avoid confusion, make sure the UI controls are distinct from scene elements. For specific details, refer to "User Interface Layout" on page 26.

Similarly, limit the objects in your scene to those required for the task. Display objects at the appropriate level of detail for their placement in the scene. Remember that objects that are farther away in the real world appear both smaller and in less detail.

Chapter 3 Design Guidelines

The design guidelines in this chapter cover a variety of topics about both designing both the 2D and 3D parts of your application. Refer to the following sections for details:

User Input

The zSpace system comes with a stylus that is designed specifically for user input in a stereoscopic 3D environment. It can move in three dimensions, point at different angles, and is not constrained to zero parallax. In general, this is a more natural way to interact with the zSpace display than a mouse, which is designed for a 2D interface. However, if you are porting an application that relies on mouse and keyboard input, you may consider having the stylus emulate the mouse or having the mouse emulate the stylus.

The zSpace system supports any input device that adheres to the USB 2.x specification. This section covers design guidelines for the stylus, the mouse, the trackball, the keyboard, as well as using multiple input devices. Note that in all cases, you want to avoid disruption by forcing the user to switch input devices.

> **Note:** Regardless of the input device, your user interface should minimize the number of clicks or keystrokes required to complete an operation.

For more information on this topic, you can watch the **zCon 2013 presentation on input devices**. Note that this presentation is more technical and it includes demonstrations. However, the presentation does not focus on design guidelines.

Using the Stylus

There are several different aspects of the stylus to consider:

- Stylus controls
- Visual representation in the display
- Mouse emulation

Stylus Controls

The stylus includes a forward center button and two left/right buttons. By default, the primary button is the forward button, with the right and left buttons defined as secondary and tertiary. However, just like on the mouse, the user can remap the buttons as needed.

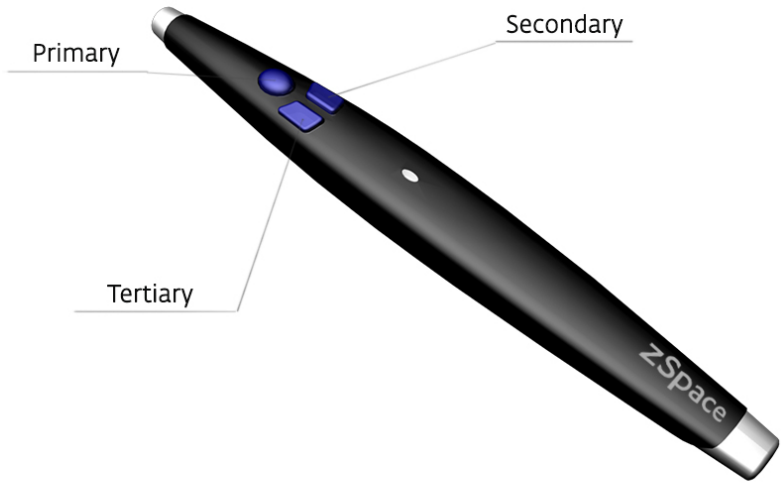

Figure 3-1 The zSpace Stylus

We recommend your application use the buttons as follows:

- Primary button: select objects and UI controls
- Secondary button: open context menus
- Tertiary button: perform other application-specific functions

These mappings match the default mouse behavior, which will make it easier for your users to learn the stylus buttons' functions.

> **Note:** All functionality should also be available to the user via a tool palette or menu options. While expert users might use all three stylus buttons and keyboard shortcuts, you should provide an easier method for novice users.

If your application does not use all three buttons, you must decide how to handle the user pressing a non-functional or unmapped button. If nothing occurs, users are apt to think the stylus is broken. For a very simple application that uses only one button, you can map all three buttons to the same function. Alternatively, you can give feedback, such as a brief stylus vibration, if the user presses a non-functional button.

The stylus includes two forms of feedback: vibration and an LED light. Consider whether using the stylus vibration will enhance your application's usability or increase the user's experience of immersion in the application. For example, a brief stylus vibration can give a subtle cue when the stylus hovers over an object.

The LED light can be set to red, green, blue, or a combination. For example, if you set all three colors to on, the LED displays a white light. Note that the meaning of the light may not be clear to your user.

Visual Representation of the Stylus

Much like a mouse pointer represents the mouse location, it is important to show the stylus location as well. However, this is more complicated than a 2D mouse pointer because objects in the zSpace display can appear at different depths. Your application needs to display a visual representation of the physical stylus. The zSpace UI Toolkit will provide two examples of a virtual stylus for you.

The virtual stylus includes a beam and a tip.

- The beam extends from the physical stylus along the stylus' orientation. The beam can be a fixed length or an adaptive beam that truncates when it intersects an object.
- The tip indicates the point of interaction with objects.

One of your most important decisions will be whether to use a fixed-length beam or an adaptive beam. Both have their advantages and disadvantages.

Fixed Beam: The fixed beam's length is constant. If it touches an object closer than the end of the beam, it does not interact with that object. This allows the user to pass the beam through one object and select one that is partly hidden. Use this visual representation when the user requires precise control and the ability to select small objects in a crowded environment. You can expand the power of this beam by giving the user explicit control over the beam length and size of the stylus tip. This stylus beam has a longer learning curve.

If you choose a fixed-length beam, you can implement volumetric selection. With volumetric selection, the user can drag to create a 3D volume for selecting multiple objects.

Adaptive Beam: The beam's length is infinite until it intersects an object, then it adapts and truncates to the distance to the object. This allows easier object selection, as long as precise control is not required. It is also easier to select objects that are deep in the scene. Use this visual representation when the user is casual or requires a quick learning curve.

You can include both a fixed beam and adaptive beam virtual stylus in your application. Some tools work best with one type over another. For example, the camera path tool requires a fixed-length beam because the user must place points at a specific location. On the other hand, the selection tool can work with either an adaptive or fixed-length beam. You can also let the user switch between the fixed beam and adaptive beam virtual stylus.

Regardless of the stylus beam, you should maintain alignment between the physical stylus and its visual representation in the display, so that they appear as one. Additionally, you should maintain a consistent scale of movement between the physical stylus and its visual representation. Given a 1:1 scale between the stylus and its representation, if the user moves the physical stylus 10 cm in the x, y, and z axes, the visual representation should move 10 cm along the same axes. The same is true for rotation of the physical stylus.

> **Note:** Depending on your application needs, you might design a different representation of the stylus than a tip and beam. For example, as described below, you can use a standard mouse pointer.

Mouse Emulation

If you are porting an existing application into the zSpace system, your application most likely includes a lot of mouse interaction. You can define the stylus to emulate mouse behavior so that users do not have to switch between the stylus and the mouse. For mouse emulation, we distinguish between your existing, or legacy, 2D UI controls and new zSpace-specific 2D UI controls. The legacy UI controls are standard Windows controls. Users who are already familiar with your 2D version of the application usually use a mouse to select these controls.

We provide the following guidelines for using the stylus to emulate the mouse:

- Over the scene, represent the stylus as a beam with a tip.
- Over the zSpace 2D UI controls, represent the stylus as a beam with a tip.
- Over legacy 2D UI controls, display the stylus as a mouse pointer instead.
- In cases where the legacy 2D UI is next to the scene or the zSpace 2D UI, use distance and stylus orientation to distinguish between stylus behavior and mouse emulation. Display a beam when the stylus is held at a distance from the screen, changing to a mouse pointer when the stylus moves closer.

For mouse emulation, we recommend that you map the stylus buttons to the native OS settings for the mouse.

> **Note:** If your application supports both the stylus and mouse as input devices, it should display only one pointer at a time. For example, when the user switches from the mouse to the stylus, remove the mouse pointer from the display, so that only the stylus beam appears.

Using the Mouse

Although we believe the stylus is a natural tool for zSpace, the mouse may be better in some situations. For example, if you are porting an existing application into the zSpace system and your users have a lot of muscle memory associated with the mouse, it might make sense for the mouse to emulate the stylus.

The key challenge for the stylus emulation is changing depths with the objects in the scene. The mouse pointer should always be on top of objects, which can appear at any depth. This section discusses two specific use cases for stylus emulation:

- Avogadro is an application that models molecular structures. Users primarily select and view content.
- Autodesk Maya® is a 3D animation application. Users need to both select and create content.

In both cases, zSpace has created a plug-in that displays content in a stereoscopic 3D viewport. The mouse pointer is displayed in stereoscopic 3D within the viewport and appears as a standard 2D mouse over the rest of the user interface. Neither implementation displays a stylus beam, but both add a z coordinate to help the mouse pointer move at different depths. The differences in the implementations are based on the application requirements.

Avogadro's users do not require a great deal of precision. Thus, the implementation is conceptually like an adaptive beam that truncates to hover over the nearest object. The angle of the beam, if it was displayed, would be drawn from the user's head through the mouse's x, y coordinates until it intersects an object. In other words, head tracking provides the orientation of the invisible beam, which supplies the z coordinate. The mouse pointer automatically rises and falls as it encounters objects at different depths.

Note: In the Avogadro implementation, the z coordinate is passive. The implementation uses the z coordinate for drawing the pointer, but it uses only the x,y coordinates for object selection.

Maya's users require more precision for drawing. A second concern is that Maya has an extensive 2D user interface that surrounds the viewport. When the mouse interacts with objects in the stereoscopic 3D viewport, the "stylus beam" must be contained within that 3D volume. In other words, it should not accidentally hover over the surrounding 2D UI. For this reason, the invisible beam is at a fixed perpendicular angle to the screen. This implementation is conceptually similar to a fixed-length beam. Users can change the length of the beam as needed, so they are manually setting the z coordinate.

Note: Using head tracking for the z coordinate would introduce some drifting based on small head movements. This would not meet Maya's requirements for precise control.

Recall that in "Using the Stylus" on page 19, we noted that an adaptive beam is easier to use, but less precise. The fixed-length beam is more precise.

In general, we recommend your stylus emulation perform in the following ways:

- Display the mouse pointer in stereo when it is within the 3D viewport. This gives the mouse pointer the appearance of depth. When it is over a 2D UI, display a standard mouse pointer.

- Display the mouse pointer at zero parallax when it enters the scene, until it touches an object. At that point, move the mouse pointer to the plane of the object. Keep it on that plane until it touches another object on a different plane.
- Keep the mouse pointer's size constant as it moves on different planes. Normally, an object's size changes with its depth in the scene. However, if you make the mouse pointer smaller as it moves deeper into the scene, it will be harder to use.

Using the Trackball

In our experience, the trackball can be very powerful as a secondary input device. The user holds the stylus in the dominant hand and the trackball in the non-dominant hand. For example, in the Avogadro application, a user can select an atom with the stylus and use the trackball to rotate the molecular structure around the selected atom. Although the rotation is possible with the stylus alone, the wrist does not move as freely as the trackball.

We have also tested the trackball in other applications. The trackball works well in these situations:

- While one hand clicks and holds an object with the stylus, the other hand rotates the object with the trackball.
- When no object is selected, the trackball rotates the entire scene.
- If the trackball includes a scroll wheel, the scroll wheel can support a zoom feature.

In general, for ease of use, scale the rotation to be 1:1 with the trackball motion. For example, if the user rotates the trackball 180 degrees from right to left, then the object or scene should also rotate 180 degrees from right to left. Similarly, rotating the trackball from front to back should produce the same amount of rotation in the same direction.

Note that some applications will work better with a different scale. For example, if precision is required, try scaling the object rotation at 1:10 with trackball motion, where 10 degrees of trackball rotation causes only 1 degree of object rotation. User testing will help you determine the appropriate scale.

There are some technical issues concerning trackballs. Windows reports a trackball device as a mouse. If the user has both a mouse and a trackball, both devices can control the mouse pointer. To avoid these conflicts, we recommend the following:

- When the trackball is in use, the application hides the mouse pointer.
- When the mouse is in use, the application displays the mouse pointer.
- The application offers a keyboard shortcut to switch from the trackball to the mouse.

  When the trackball is active, we do not recommend using a click of the standard mouse to switch devices. In our experience, it is too easy to accidentally reach for the mouse when you intend to use the trackball.

For more general guidelines about using two devices at the same time, refer to "Using Multiple Input Devices" on page 26.

Using the Keyboard

Your application should provide standard Windows keyboard shortcuts. Based on the types of behavior that are likely in a zSpace application, we suggest the following:

Key	Meaning
F1, Shift+F1	Display help. At this time, we do not offer specific guidelines on how you should implement help.
F5	Refresh the active window in the user interface.
Tab	Move between the application control bar, palette, and inspector when one of these has focus. Refer to "Application-Level Controls" on page 27 for more information on these controls.
Ctrl+A	Select all. Whether this applies to in-scene objects or a UI control depends on the current focus.
Ctrl+C, Ctrl+Insert	Copy the selected object or item.
Ctrl+X	Cut the selected object or item.
Ctrl+V, Shift+Insert	Paste the last object or item from the clipboard.
Ctrl+Y, Alt+Shift+Backspace	Redo
Ctrl+Z, Alt+Backspace	Undo
Ctrl+P	Print
Ctrl+S	Save
Shift+F10	Display the context menu for the selected object or item.
Alt+F4	Close the application.

Note: Your application may not support all of the above operations.

We also recommend keyboard shortcuts for options in your 2D UI and any 3D controls you implement.

You can also provide keyboard shortcuts to change the stylus behavior. You could give users control over the length of a fixed stylus beam or provide shortcuts to change modes, such as from reposition to rotate.

Using Multiple Input Devices

As stated in the beginning of this section, you should minimize the user's need to switch between input devices. However, depending on your application's complexity, you may choose to support two input devices, one in each hand. The input devices could be used at the same time.

Where possible, it is desirable to offer both novice (stylus only) and advanced (stylus + keyboard) options. For example, novice users might explicitly control the behavior of the stylus with a menu option, while advanced users might combine the stylus with keyboard options. This is very similar to how most users start with 2D applications, using the mouse initially, then gradually using keyboard shortcuts for frequent tasks. You can leverage common uses of the Shift and Ctrl keys such as multiple selection and fixed scaling. The Enter button often has an "execute" meaning.

You can also combine the stylus with a trackball, as described in "Using the Trackball" on page 24.

Yves Guiard proposed the following guidelines[1] for two-handed tasks:

- The dominant hand uses one device for fine-grained control, while the non-dominant hand uses another device for gross manipulation.
- The device in the non-dominant hand controls the viewpoint.
- The device in the dominant hand starts the manipulation.

User Interface Layout

This section discusses the zSpace Design Philosophy and the different types of application-level controls.

zSpace Design Philosophy

As we stated in the beginning of the "Fundamental Design Principles" on page 13:

> We believe some things are better displayed in 2D, based both on complexity and the users' past experiences with 2D interfaces. For example, it is far easier to read text in 2D than in stereoscopic 3D. Users will be more efficient with user interface controls in 2D because the physical interaction is less complex in two dimensions.
>
> However, you will be designing your application's scenes and objects in stereoscopic 3D. In many cases, it is far more powerful to model the real world and leverage users' real world experience in 3D. This taps into our ability to understand certain things better in 3D. For example, we can more readily experience and understand spatial relationships and physical constraints in 3D.

The underlying philosophy for designing zSpace applications is to use 2D where it offers more value than 3D. Use 3D where it offers advantages over 2D.

---

[1] Guiard, Y. (1987). Symmetric Division of Labor in Human Skilled Bimanual Action: The Kinematic Chain as a Model. *The Journal of Motor Behavior* 19(4): 486-517.

Application-Level vs. Scene

As we think about a user interface in the zSpace system, first we distinguish between application-level controls and the application's scene or content.

Application-level controls are top-level functions such as exit, file management, and navigation. For your application-level controls, your application can benefit from the advantages of 2D, placed at zero parallax:

- Text is easier to read in 2D.
- Viewing comfort is maximized at zero parallax.
- UI controls are easier to select at zero parallax.

Note: When we refer to 2D for application-level controls, we mean that your application-level controls should not be full stereoscopic 3D objects with interaction on multiple faces. However, your application icons can have a subtle 3D appearance and still be easy for users to select.

The application-level controls are the equivalent of a HUD (head-up display). Like a HUD in a computer game, the controls may either be permanently displayed or temporarily displayed as needed.

The scene is the core of your application, containing your 3D content. For your scene, your application should take advantage of the strengths of the zSpace system: a realistic stereoscopic 3D display, complete with head tracking and direct manipulation via a stylus.

Application-Level Controls

The zSpace Design Philosophy defines three types of application-level controls: an application control bar, palette, and inspector.

The **application control bar** is the application's main control and contains all of the application-level commands. It should contain commands for file operations, preferences, and exiting the application. The application control bar sits at the bottom of the screen. For an example of this placement, refer to the zSpace Concepts application.

The **palette** contains stylus tools, or modes, such as selection, scale, and the camera path tool. The palette appears on the right side of the screen.

The **inspector** shows the details of an object. The inspector's contents updates as the user changes an object in the scene. The inspector also appears on the right side of the screen.

All applications require the application control bar. We believe that most applications will also need a palette, but only some applications will require an inspector.

Note: Whether or not your application needs a palette or inspector, you can also use context menus in your user interface. A later version of these guidelines will include specific recommendations.

*Figure 3-2 **Layout of Application-Level Controls***

The above figure shows the placement of the application-level controls. It is not intended to show the look and feel of these controls.

Place the application control bar, palette, and inspector at zero parallax. As noted earlier, your application can either display these types of controls permanently, show them as needed, or allow the user to slide them in and out of place.

Left and Right-Handed Layouts

The suggested layout of the palette and inspector are best for right-handed users. With the controls on the right, the user can reach them more easily and the controls are not blocked by the position of the user's hand and arm.

We believe this layout should be configurable for left-handed users so that the palette and inspector are placed on the left instead. Make this configuration available via a Preferences option on the application control bar. When the user selects the left-handed layout, offer the user the option to update the stylus settings on the zSpace Control Panel as well.

Content Layout

This section provides design guidelines for creating scenes, which contain your application's content. The content is separate from the application-level UI controls. For guidelines on UI controls, refer to "User Interface Layout" on page 26.

Content Creation

Content creation is a complex process, starting with a 3D modeling application and ending with the model displayed by the rendering engine you choose. If you do not have experience with 3D modeling, we recommend you hire someone or allow for time to learn.

Generally the modeling process will include the following steps:

1. Build – or purchase – a 3D model.
2. Add materials to the model. You can create or purchase these.
3. Apply or adjust shaders to get the desired surface quality.

Depending on your application requirements, you might start with polygon models, Non-Uniform Rational Basis Spline (NURBS) surface models, or parametric solid models. The zSpace system does not restrict the type of 3D model you use.

Next, depending on the rendering engine you use, you may need to convert the model from one format to another. For example, Unity requires polygon models. However, you could create NURBS models in one modeling application and then convert them to polygons in another. You can use any rendering engine that is zSpace-enabled.

Usually you will set lighting within the rendering engine.

Finally, you will probably need to make adjustments to your model for optimum results in your rendering engine. Some rendering engines support certain changes in their editors. In some cases, you will need to adjust the model in the modeling application.

The following sections describe some things that contribute to your model's effectiveness:

- "Model Performance" on the next page
- "Ghosting and Flickering" on the next page
- "Realism" on page 31

Model Performance

Your scene will need to refresh fast enough for users to experience smooth animation and acceptable head tracking. We have found that a 60 frames per second (FPS), 30 FPS per eye, is the minimum for comfortable viewing. For long sessions, we recommend higher rates. Because the zSpace monitor refreshes at 120 Hz, the maximum possible FPS is 120, or 60 per eye.

Your application's desired FPS may vary, depending on your application requirements. You will need to balance the desired resolution against requirements for smooth animation.

> **Note:** Although model complexity can affect your application's refresh rate, it is not the only factor. For more information, refer to "Improving Performance" on page 40.

Ghosting and Flickering

In the zSpace display, the scene is constantly redrawn based on the position of the viewer's head, with separate images for the left and right eye. If each eye sees an image meant for the other eye, this can cause ghost images. In other words, the user sees a faint duplicate image.

To eliminate ghost images, follow these guidelines:

- Avoid using strongly contrasting colors, hues, values, and patterns.
- Use textures and variable brightness instead of solid colors.
- Try to match the average color value of the objects to the average color value of the background.

Patterns next to solid-colored objects may have ghost images. In addition, evenly-lit geometric patterns are also subject to ghosting. Patterns with soft transitions work best.

Note that the left and right-eye images at zero parallax are rendered on top of each other, so ghosting cannot occur. Although you can use more contrast at zero parallax, ghosting can occur on objects in front of or behind the object at zero parallax. In addition, if the user moves the object out of zero parallax, ghost images will appear if it is in high contrast.

Sometimes you cannot modify a model to reduce ghosting. For example, your application may be showcasing products; you do not have the freedom to adjust colors and textures. In such cases, you will have to adjust the background to decrease ghost images.

In addition to ghost images, users may also experience flickering. Thin lines have the potential to cause flickering, whether in text, objects, or a border. As the user moves his head or the object moves, pixels on the line may not be displayed. This is more noticeable with high-contrast objects. The guidelines for avoiding ghost images will also eliminate flickering.

Realism

One of the strengths of a stereoscopic 3D environment is that objects can more closely match the real world. However, you should also consider the user and the task. Do not attempt to recreate everything photo-realistically just because you can. In some cases, a cartoon or a pencil sketch may be a better alternative. Consider the following examples:

- Photorealism would be important in an application for medical students. An anatomy game for children could easily use cartoons instead.
- Faces rendered in stereoscopic 3D may look artificial or trigger the “uncanny valley” effect.[1] A sketch of a well-known person can be easily recognized and avoid the risk of appearing “fake.”
- A sketch can represent a storyboard or concept that is still in the planning stages.

When realism is your goal, make sure your models have the same properties that they do in the real world. For example, reflective objects should appear to reflect light, while glass should appear transparent. Include light sources and shading.

> **Note:** You need to balance realism against performance. Some aspects of realism, such as reflections, can be costly in terms of computational load.

Be selective about how much detail and how many objects to place in the scene. If something is not important for the task, do not show it. Display the key items in the level of detail that makes sense for your application and your user. For example, whether to hide or show the wiring in a house schematic depends on the purpose of the schematic.

Content Placement

Content placement refers to both the depth in which you place objects and how you scale them. Refer to the following sections for details:

- "Depth" below
- "World Scale" on page 34

Depth

Recall from "Experiencing 3D in zSpace" on page 8 that objects in positive parallax appear behind the screen or inside the monitor, while objects in negative parallax appear in front of it. Within those regions, there is an optimum range for comfortable viewing. When objects are too close, the user has trouble focusing and will see

---

[1] The uncanny valley hypothesis refers to viewers feeling revulsion when a human likeness is almost, but not perfectly, realistic. A less realistic representation evokes a more comfortable response. This has been observed in robotics and 3D animation.

double images. While viewing distant objects is not uncomfortable, at some point depth is no longer discernable. Stereoscopic vision is a primary cue for objects that are relatively close to the viewer, becoming less of a factor as distance increases.

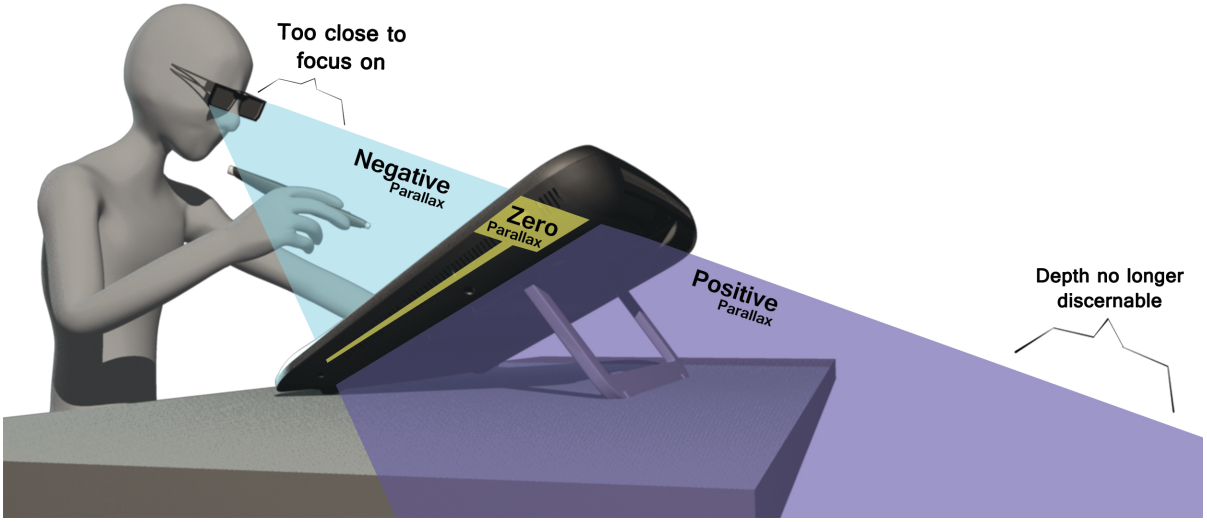

Figure 3-3 Viewing Comfort

Clipping planes control the field of view's depth as follows:

- The **near clipping plane** defines how close to the user objects may appear. If the user attempts to move an object closer, it is not rendered or is only partly rendered.
- The **far clipping plane** defines how far from the user objects may appear. Beyond the far clipping plane, objects are not rendered or only partly rendered.

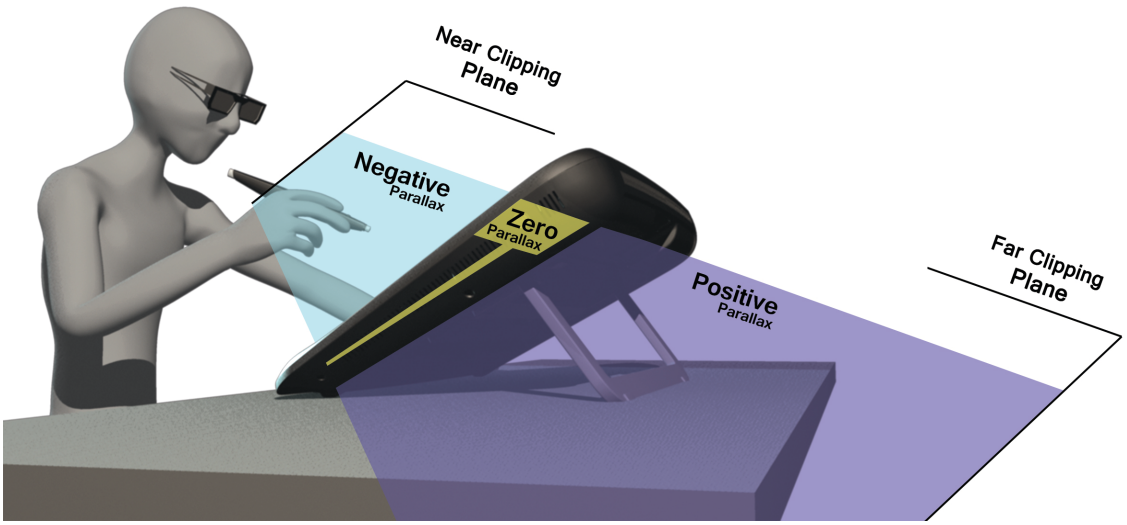

*Figure 3-4 **Clipping Planes***

You can adjust the clipping planes with the zSpace SDK's Core libraries.[1] Follow these guidelines for setting your clipping planes:

- Set the near clipping plane to greater than half the interpupillary distance. By default, the interpupillary distance is 0.06m.
- Make the ratio between the far and near clipping planes as small as possible. This will improve the numerical precision in the rendering algorithms. When the ratio is very high, such as 1000:0.001, overlapping objects can appear in the wrong order. A more typical ratio is 100:0.1.
- Set the clipping planes far enough apart to accommodate all the relevant content in the scene.

For example, in the zSpace Experience application, the near clipping plane is 0.1 and the far clipping plane is 1000.

Generally, the space defined by the clipping planes should be at least as large as viewers' comfort zone. Note that the comfort zone decreases as the viewpoint moves, so that focusing on near and distant objects can become uncomfortable for the viewer.

For information on other adjustments you can make with the zSpace SDK, refer to "Stereo Optimization" on page 45.

[1] zSpace's Core libraries are available in both a native zSpace SDK and in a Unity-specific SDK.

As you design your application, you must decide where to place objects in your scene. Generally, people are more comfortable with positive parallax than negative parallax. In addition to viewing comfort, consider the user's physical interaction with objects in the scene. It is harder to manipulate objects with the stylus if you place them too far away.

World Scale

Sometimes an application requires a greater range of distances than you can comfortably view within the clipping planes. To resolve this problem, you can adjust the world scale. As an example, if your world is 100 times larger than the zSpace display, use a world scale of 100. In that case, objects that appear to be 500m away are actually rendered much closer and remain easily viewed. For example, if you have a large model of the solar system, you would need a large world scale. The actual world scale would depend on the size of your models.

On the other hand, if your application needs to display microscopic detail, you would set a smaller world scale so objects appear larger than in the real world. Choosing the correct world scale depends on what you are modeling. When it is feasible, we recommend using a 1:1 world scale to provide the most realism.

Note: In addition to adjusting the world scale, you may also need to adjust the viewpoint to produce the desired effect.

Chapter 4 Design Considerations

This chapter covers a variety of topics that you should consider when designing your application. In general, we do not offer concrete guidelines because the correct approach depends on your specific application. Refer to the following sections for details:

Avoiding Application-Level/Scene Conflicts

A key challenge will be preventing your application controls – the control bar, palette, and inspector – from interfering with your in-scene objects. This can be a difficult problem to solve, and the best solution will depend on your specific application. This section offers design considerations, rather than absolute guidelines.

> **Note:** We would like to hear from the zSpace community regarding your experiences with this issue.

There are at least six different solutions:

1. Reserve space for your application-level user interface
2. Temporarily move the viewpoint
3. Temporarily scale the scene
4. Limit the scene to positive parallax
5. Temporarily hide the in-scene objects
6. Temporarily tunnel through the in-scene object

We think the first three approaches are the most viable, but that the last three approaches have significant problems.

The following subsections discuss the strengths and weaknesses of each option.

Best Approaches

This section discusses the three most viable solutions.

Reserve Space for Your Application-Level Controls

In this approach, you set aside a portion of the display for the application control bar and, if necessary, the palette and inspector. The advantages are that it is a relatively simple solution and it provides the user with a sense of stability. You can either set aside a 2D space at zero parallax or a complete 3D volume.

- A 2D space may be sufficient if the user has complete control over the placement of objects in the scene. If an object is above or below the control bar, the user can move it to access the control bar. zSpace used this approach in its zSpace Concepts demo application.
- A 3D volume ensures that in-scene objects can never obstruct the control bar and that the control bar cannot hide parts of the scene. However, this leaves you with a smaller area for your scene.

Temporarily Move the Viewpoint

In this approach, you gradually move the viewpoint so that all objects are in positive parallax, leaving room for the user interface. Your application must clearly notify the user that the objects themselves are not being modified. When the UI controls are no longer required, your application gradually resets the viewpoint back to its previous setting.

This approach maximizes the area available for the scene until the user interface is needed. However, moving objects for the user violates the principle of user control. For more information, see "User Control" on page 15.

Temporarily Scale the Scene

In this approach, you gradually shrink the entire scene to make room for the control bar, palette, or inspector. As in other examples that take action for the user, it is important to let the user know that objects are not being edited.

This approach has the same advantages and disadvantages as moving the viewpoint. In addition, the entire scene must scale small enough to fit in the allotted space, but still be large enough for the user to easily interact with it.

Less Desirable Solutions

We feel these approaches are less likely to work.

Limit Scene to Positive Parallax

In this approach, your scene is never in negative parallax. That is, it appears behind the screen, while your user interface is always at the screen's surface.

This solution gives you a lot of space for your user interface, but less space for your scene. This also prevents your users from bringing objects into negative parallax to examine them more closely.

Temporarily Hide Conflicting In-Scene Objects

When you need to display the control bar, palette, or an inspector, you can temporarily hide any in-scene objects that interfere with it. This should occur gradually so the user can see that the objects are not arbitrarily disappearing.

The advantage to this approach is that it maximizes your 3D space. However, this will not work if the user interface is required to interact with the now-hidden object.

Tunnel Through the Objects

Instead of completely removing in-scene objects that block the application-level UI, you can temporarily hide portions of the objects obscuring it. This is like drilling a tunnel through the scene. The scene should be redrawn gradually rather than immediately removed. This has the same advantages and disadvantages as the previous approach.

Navigation

Almost all applications require navigating from one place to another. This could be navigation within the scene or navigation between different scenes. Good navigation tells users their current location and lets them go where they want to.

Unless your application displays all its contents initially, the application must convey its navigation model to the user. Even in applications with a single scene, the user may still need to navigate within the scene. Except in games that include discovering hidden clues, navigation must be easily understood. You can use tooltips or callouts to label navigation techniques, giving the user the option to hide them when no longer needed.

2D Navigation Metaphors

Borrowing from 2D user interfaces, you can use these familiar navigation techniques:

- Wizards move you through the steps to perform a task.
- Breadcrumbs indicate where you are within an application and usually provide a way back to a prior location.
- Hyperlinks provide a way to jump from one location to another.
- Trees or outlines provide a structured view of all the hyperlinks available, such as the Table of Contents in this document.

3D Navigation Methods

The zSpace Experience demo application includes a portal, or teleportation, as a means of navigation. This is the equivalent of a hyperlink that lets you jump from one scene to another.

Another common method of navigation is travel, in which the user changes the viewpoint through motion. Computer games include many examples of flying and walking. For navigation by flying or walking, you will need to decide both how to control movement and how to restrict movement. Controlling movement includes considerations such as direction and speed. You can restrict movement to specific paths or allow movement in any direction within a restricted space.

As you design your application, think about your user's goals and knowledge. Does the user want to explore and gather information while moving from one location to another? Or is the goal to navigate to the destination as quickly as possible? Does the user know the desired destination and how to get there? Is a search mechanism required? These questions are particularly applicable if your application has large scenes or multiple scenes.

Real World Physics

When you design your application, you need to decide whether to implement real world physics and to what extent. Incorporating physics into your application can include:

- Collision detection
- Rigid-body and soft-body dynamics
- Fluid dynamics
- Motion control

For a quick visual introduction to different types of physics, you can view a video on this **Wikipedia** page: **Physics Engine**.

> **Note:** Implementing physics can be expensive in terms of computational load. You will need to balance the value of implementing physics with the application's performance.

This section describes different categories of physics that you might include in your application.

Collision Detection

The purpose of collision detection is to determine if objects might touch or intersect each other. For some applications, this can be critical. You can define bounding boxes or spheres as collision meshes for your models. The collision mesh defines the boundaries at which objects may not intersect.

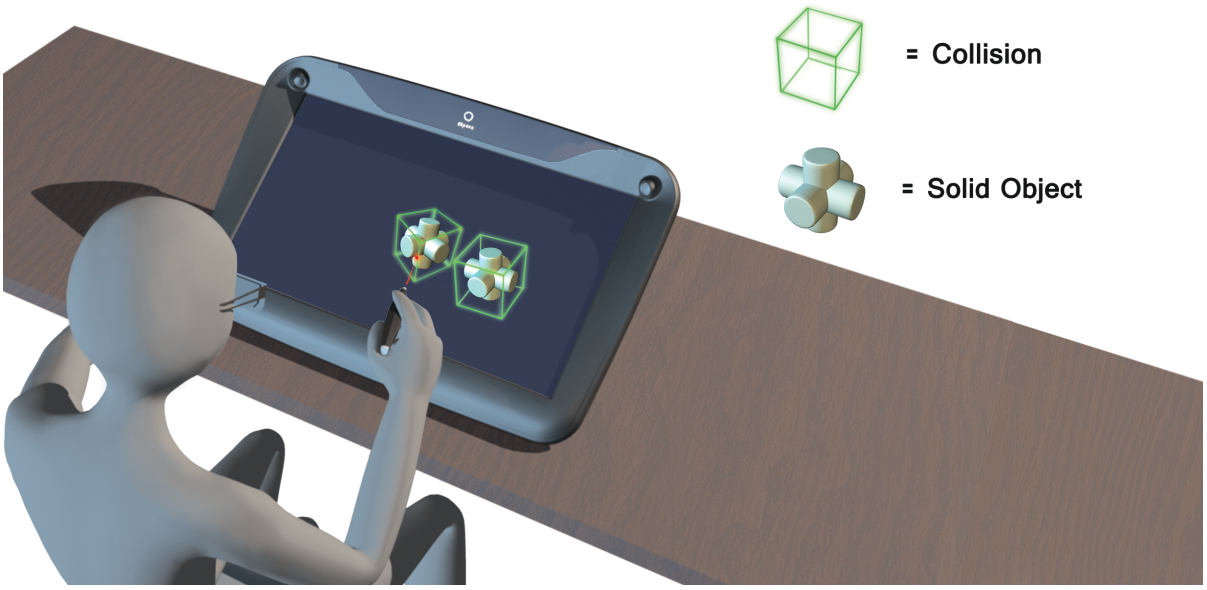

*Figure 4-1 **Collision Meshes***

If the user's tasks do not require precision, you can define a simple bounding box or sphere. For example, you could define gross boundaries for a chair in a furniture layout application. However, for a mechanical assembly, the objects probably require more precise boundaries.

You can see an example of collision detection in the zSpace Concepts demo application. When the Physics option is on, objects cannot intersect one another. You can also turn the option off, which allows objects to move through one another.

Dynamics Simulation

You can use dynamics simulation to define whether your objects are rigid, soft, or fluid. If two rigid objects collide, neither would change shape. If you model a soft object, such as foam, then its shape should change as a result of impact. Modeling water, complete with fluid dynamics, is even more complex.

If your application requires dynamics simulation, determine how much accuracy is necessary. For example, consider the behavior of dropped objects (pianos, anvils) in a cartoon. Contrast that with the importance of modeling physical behavior in a simulation for building demolition. In the latter case, precision is important to accurately model the results.

Motion Control

Motion control defines how an object moves. Gravity is one aspect of motion control. As mentioned earlier, the zSpace Concepts application includes a Physics option. When Physics is off, gravity is disabled and objects hover in mid-air, making them easy to inspect. When enabled, the objects suddenly drop to the floor.

If inspecting objects is a key aspect of your application, implementing gravity would actually make your application harder to use. On the other hand, if your application simulates car crashes, you would implement gravity, as well as other aspects of an object's behavior: whether it bounces, how it rolls, and so on.

Performance and Responsiveness

First it is important to distinguish between performance and responsiveness:

- Performance refers to the system performance, measured by CPU speed, frames per second, and so on.
- Responsiveness affects the user's perception about performance. This measures response time, or how quickly the application responds to a user's actions, not how quickly the application completes the actions.

Improving Performance

This section defines performance based on frames per second (FPS). A low FPS rate can reduce or eliminate the benefits of head tracking. When the scene refreshes too slowly for the user's head movements, this is worse than no head tracking. As stated in "Model Performance" on page 30, we recommend your scene refresh at 60 FPS (30 FPS per eye) at a minimum. To improve performance, you need to consider both the computational complexity of your scene and your application logic.

Checking the Scene's Impact

Your models' polygon count, the resolution of the textures, and other rendering information, such as shaders, contribute to the computational load. If your application's performance is slow, we recommend you follow these steps to identify the problem:

1. Check lighting first, disabling various options to see if any single change has a large impact:
    - The number of lights
    - The type of lights such as directional and point lights
    - Lighting effects such as shadows and bounced lights
2. Check the shaders, disabling various options to see if any have a large impact:
    - Transparency
    - Specular

- Refraction
- Reflection

Although you cannot disable a diffuse shader, you can replace the associated 2D texture map with a single color.

3. Check the textures' size. Larger or higher resolution textures can slow performance.
4. Take a look at the polygon count.
5. Finally, consider whether you simply have too many objects in the scene.

If performance problems persist, you can try to improve performance by using lower level-of-detail (LOD) models or trying lower resolution textures. You can also allow make allowances for lower quality graphics cards by dynamically choosing lower LOD models. In some cases, it makes sense to let the user choose the settings for rendering quality.

Checking the Application Logic

For application logic, make sure your foreground or rendering thread consistently has low enough latency to allow the renderer to maintain a high frame rate. Avoid heavy processing in these threads. Because stereoscopic 3D applications have to render the whole scene twice, they require much more graphics processing power than 2D applications. Any optimizations you can make to your application will help reduce the load on the graphics processor.

Improving Responsiveness

Regardless of the absolute system performance, you can improve users' perceptions by providing immediate feedback.

- For your UI controls, you can display immediate feedback by changing the appearance of pushed buttons and selected menu items.
- For the in-scene objects, you can change the appearance of objects when they are selected to indicate the user's action. You might display the wireframe or highlight the object in some way. This is a familiar convention from 2D user interfaces.

Users must receive immediate feedback about physical actions (button pushes, object selection) within 0.1 second. Otherwise, they will believe the action failed and try again.

In zSpace applications, responding quickly to the user's stylus movements is critical. It is more important to show an object's movements as the stylus moves than to show the complete object. You can try displaying wireframes or lower LOD models for faster rendering during the user's actions.

If an operation takes longer than a second to complete, display a busy cursor. This acknowledges the user's request, so he or she does not try again. For operations that take longer than 10 seconds to complete, display a dialog box or progress bar.

Using Special Effects

Use special effects when they are appropriate to the task and the audience. This section offers some guidelines about using 3D outside the scene, animation, the stylus' feedback, and sounds.

3D

In general, reserve stereoscopic 3D for your scene and objects in your scene. Use 2D for UI controls and text. Note that you can give your UI controls a subtle 3D appearance, but for easy use, do not make them true stereoscopic 3D objects that support user interaction on multiple faces.

If you are tempted to replace a control with a 3D object, consider whether it enhances your application or whether a familiar 2D control would work as well. For example, you could display a 2D slider bar or a place lights spatially to control lighting. But should you?

- If the lighting is an integral part of your application and your scene is uncluttered, lighting objects might enhance your scene.
- If the lighting is a rarely used setting, use a 2D control that is only present when needed.

Animation

Use animation when it is necessary to illustrate a process or notify a user of key information. Examples include:

- Drawing attention to an object moving on or off screen
- Familiar 2D metaphors, such as animating a file transfer

Stylus Feedback

The stylus can provide feedback through vibration. For example, you can use a very brief stylus vibration to tell the user that an object is hovered or selected. The user may interpret this feedback similar to touching an object in the real world, even if he or she does not consciously notice the vibration. In our experience, although users often do not notice the vibration, they are more likely to be confused when the vibration is absent.

Other aspects of the stylus are covered in "Using the Stylus" on page 19.

Sounds

If your application uses sounds with notifications or alerts, give the user control over whether or not they occur.

Your application may also use sounds as part of the scene. In this case, make sure the audio is synchronized with the 3D experience. Otherwise the user may experience cue conflicts, where audible information does not match the visual and physical feedback.

Universal Design

Ideally, design your application for the widest possible audience by considering the following issues:

- Internationalization
- Left vs. right-handedness
- Accessibility

Internationalization

We recommend you design your application so it can be easily modified for different locales. Follow these guidelines:

- Make it easy for developers localizing your application to replace the text associated with messages and UI controls.
- If applicable, make references to weights and measures configurable so they can be metric, U.S.-based, or British-based.
- Try to avoid culture-specific references in your scene.
- Be aware that color has specific meanings in other countries. For example, Western cultures often associate white with purity, but many Asian cultures associate white with mourning.

Left and Right-Handedness

The zSpace stylus is configurable so that users can change the right and left buttons' mapping. For more information, refer to "Using the Stylus" on page 19.

We also recommend making the placement of application-level controls configurable for the left or right side of the screen. For more information, refer to "Left and Right-Handed Layouts" on page 28.

Accessibility

For users with disabilities, careful planning can improve your application's accessibility. For example:

- Enable keyboard shortcuts and mouse options for users with limited physical mobility.
- Add audio options for visually impaired users.
- Use additional cues besides color for colorblind users.

The zSpace system provides additional possibilities for creating applications designed for users with disabilities. For example, stylus vibration might help a visually impaired user "see" that the stylus is intersecting an object. The user can then select the object and bring it closer to examine it. Alternatively, you could display an enlarged view of the object in a PIP (Picture in Picture) window.

When you design for colorblind users, you must be careful that your solutions do not cause ghost images, as described in "Ghosting and Flickering" on page 30. Depending on your application, you can try using different depths, in addition to color, to distinguish between different categories of information.

Chapter 5 Stereo Optimization

This section discusses common problems in stereoscopic 3D applications. We hope that presenting this information in these Design Guidelines helps you to avoid these pitfalls. If you are responsible for the application's UI design, but not its architecture, please review this information with your application developers.

This section discusses how to optimize stereoscopic 3D in your application by changing key parameters in the zSpace SDK's Core libraries[1]. However, if your application is using a proxy server such as VRPN instead, then you will need to understand the zSpace coordinate systems and the stereo calculations used by your application.

Applications Using the zSpace SDK

The following sections provide ways to improve your application's visual quality with the zSpace SDK. We suggest you make adjustments in this order:

1. Set the world scale.
2. Adjust zoom and wide angle effects.
3. Make small changes to either the interpupillary distance or stereo level.
4. Make small changes to the zero parallax offset.

Adjusting Model Size with World Scale

The zSpace system uses real world units, where 1 unit of measure equals 1 meter. If your application's scene is too large or too small, it will not fit into the zSpace display space correctly. If the scene is too small, your application loses the stereoscopic 3D effect and head tracking does not offer any benefit. If the scene is too large, it is uncomfortable to view and small changes in head movement cause drastic changes in the view.

Although you could simply resize everything in your scene, you would also need to reposition everything. In addition, you could see a loss of precision due to floating point rounding errors. Instead, we recommend that you adjust the world scale with the zSpace SDK's Core libraries, as follows:

- For a large scene, such as the universe, increase the world scale.
- For a small scene, such as a molecule, decrease the world scale.

[1] zSpace's Core libraries are available in both a native zSpace SDK and in a Unity-specific SDK.

If the size of your scene changes, you can dynamically adjust the world scale. For example, your application might zoom from a scene of the earth to a scene of a city.

World scale adjusts the size of the display and the relative head tracking for that size display. It also effectively changes the near and far clipping planes, so you may need to adjust them after changing the world scale. In addition, you may need to adjust the viewpoint, so that objects are distributed comfortably in positive and negative parallax.

Note: The zSpace SDK does not apply world scale to the stylus, so your application will need to apply it to the stylus pose.

Using Correct Zoom and Wide Angle Techniques for zSpace

Traditional methods of simulating zoom and wide angle techniques often do not work correctly in a zSpace application because the zSpace SDK uses a field of view defined by the eye position and the physical position of the screen. You should avoid these techniques:

- Moving the camera into the scene to simulate zoom or pulling the camera away to simulate a wide angle view
- Narrowing the field of view to simulate zoom or widening the field of view to simulate a wide angle view

When you move the camera into the scene, this causes the object to come into negative parallax instead of increasing in scale. Similarly, when you pull the camera away, the object moves towards positive parallax instead of decreasing in scale. Adjusting the field of view can cause issues with objects becoming skewed.

However, you can simulate the zoom and wide angle techniques by dynamically changing the world scale and adjusting the camera position. Alternatively, you can set the field of view scale in the zSpace SDK, as follows:

- For a wide angle effect, set the field of view scale to greater than 1.
- For a zoom effect, set the field of view scale to less than 1.

Note: Large changes to the field of view scale can interfere with the mapping between the physical stylus and its visual representation. For example, if the physical stylus is pointing directly towards the display, the virtual beam may appear at a different angle. We do not recommend large adjustments unless your application uses a different input device than the stylus.

Adjusting Other zSpace SDK Parameters

With the zSpace SDK you can explicitly manipulate interpupillary distance, stereo level, and the location of zero parallax. You should only make these adjustments after correcting world scale and possibly making changes to field of view scale or the clipping planes.

- **Interpupillary distance:** This represents the physical distance between the viewer's eyes. By default, this is .06 meters. If the number is too small, the stereoscopic 3D effect is weakened. If the number is too large, users will have trouble bringing the images into focus. You can make small changes, which will adjust the offset of the two images (stereo separation). However, a large change will cause problems with head tracking and the stylus.
- **Stereo level:** This directly adjusts the stereo separation. The default is 1, for 100% stereo separation. A value of 0 would completely disable the stereo effect.
- **Zero parallax offset:** This adjusts the plane that represents zero parallax. You can use this for fine-grained control, but large changes will have a negative effect on head tracking. Generally, it is better to move the viewpoint closer to where you want zero parallax to be than to adjust this parameter.

Note: The stereo effect is determined by both the interpupillary distance and stereo level, with stereo level acting as a multiplier. The interpupillary distance is a physical distance while stereo level is a scale factor. Thus, if you have a very low interpupillary distance, a stereo level of 1 will not increase the stereo separation.

For More Information

These topics are covered in the Optimizing Stereo for zSpace webinar. Refer to one or both of the following:

- Video
- Slides and transcription

Applications Using Proxy Servers

If your application gets coordinates from a proxy server such as VRPN or trackd, then you need to understand zSpace's different coordinate systems. The proxy server will provide the stylus and head position in zSpace's Tracker Space. Your application must translate these coordinates into your application's coordinate system.

For more information, refer to zSpace Coordinate Systems. You can also learn about the zSpace coordinate systems in the Optimizing Stereo for zSpace webinar, referenced in the previous section.

Glossary

Binocular vision

Binocular vision refers to seeing out of both eyes at the same time. You will see slightly different views out of your left and right eyes, which your brain fuses the images into a single three dimensional view. The ability to see the two images as a single image with depth is called stereoscopic vision.

Haptic feedback

Haptic feedback refers to providing physical feedback to the user. The zSpace stylus can vibrate to provide haptic feedback.

Kinesthesia

This is similar to proprioception and refers to awareness of the position and movement of your body parts.

Motion parallax

When you move your head, objects remain stationary, but your view shifts. Objects that were hidden may come into view and other objects may become hidden. Additionally, objects in the distance appear to move less than objects closer to you. This is a cue that reinforces depth perception.

Negative parallax

Negative parallax refers to the space that projects from the surface of the zSpace monitor towards the viewer.

Polarization

Polarization is a property of light waves, which can oscillate in more than one way. The zSpace System uses polarized glasses to filter the light that reaches each eye. When zSpace displays two offset images, one for each eye, the glasses ensure that each eye sees only a single image, which the brain fuses into one combined image.

Positive parallax

Positive parallax refers to the space that appears inside or behind the monitor. Objects displayed in positive parallax appear to be behind the surface of the zSpace monitor.

Proprioception

This is the sense of your body's position. When combined with physical movement, it provides additional depth cues for 3D perception.

Stereoscopic 3D

Stereoscopic 3D refers to displaying objects in three dimensions, with depth and perspective. This is different from displaying a 3D object in two dimensions, such as a cube in a drawing application.

Zero parallax

Zero parallax refers to displaying something at the surface of the zSpace monitor.