zSpace®

Style Guide & Best Practices

770-00057-01_RevA

Preface

Audience

This document is intended for anyone who designs, develops, or ports applications for the zSpace mixed reality platform. This may include visual and interaction designers, developers, product managers, and program managers.

Scope

This document introduces the concepts of zSpace mixed reality, best practices, technical development, and partner brand guidelines.

Conventions

New terms and concepts appear in *italics* at first mention.

User interface elements appear in **bold**.

Links to other documents or web pages use HTML style format color and underline.

Images are not numbered. Some images have descriptive titles.

Notes and callouts look like this.

Document Organization

Contents

Chapter 1. Overview

Chapter 2. Mixed Reality Essentials

Chapter 3. Best Practices

Chapter 4. Technical Guide

Chapter 5. zSpace Partners

Related Documents

For more information about zSpace learning content, see edu.zspace.com.

For more information about zSpace development, see developer.zspace.com.

zSpace Support

For help with zSpace applications and technology, see support.zspace.com.

Editors

Clifford Champion
Victor Chapel
Vince Toscano

Authors

Clifford Champion
Vince Toscano
Phil “Captain 3D” McNally
Doug Twilleager
Jerry Tu
Jonathan Hosenpud
Katherine Loe
Bien Manual
Michael Albers

Drawings

Helen Zhu

Special Thanks

Special thanks to Phil “Captain 3D” McNally for comfort concepts and original drawings.

Special thanks to the zSpace leadership team.

zSpace source: zSpace Developer - SDK Guide Introduction (Getting Started), zSpace SDK - Native Programming Guide, zSpace SDK - Unity 3D Programming Guide, zSpace Hardware, zSpace System Requirements, zSpace Supported Graphics Cards (GPUs), zSpace 100 and 200 Control Panel and System Check walkthrough for System Software 4.0.1 and below, zSpace 300 and HP Zvr Control Panel and System Software 4.3 and above, zSpace Demos and Games, zSpace App Manager download and release, zSpace Downloads, zSpace Unity Plugin Releases.

Contents

1. Overview
zSpace Basics
- Standard Components of the zSpace System
  - Stereoscopic 3D Display
  - Eyewear
  - Stylus
  - Mouse
  - Keyboard
  - Plus Bring Your Own Devices

2. Mixed Reality Essentials
Introduction to Stereoscopic Imagery
- Human Vision
- zSpace Vision
- Focus and Convergence

Designing for Comfort
- Coupled and Decoupled Zones
  - Comfort Zone
  - Background Decoupled Zone
  - Foreground Decoupled Zone
  - Displaying Objects in the Decoupled Zone
- Window Violations
- Ghosting
- Depth Changes
- Depth Cue Consistency
- Performance Requirements

3. Best Practices
User Interaction
- Stylus Behavior
  - Direct Interaction
  - Visualization and Length
  - Rotation
  - Buttons
  - Selection
- Context Menus
- Text and Legibility

- Audio
- Haptics
- Visual Feedback
- Permanence and Magic
- Broadened Demographics
  - Handedness
  - Motor Skills
  - Color Blindness
  - Undo-ability
- 2D User Interfaces

Using More Input Devices
- Mouse
  - Stylus-based Mouse Emulation
- Trackball
- Keyboard
  - Keyboard Commands

zView Augmented Reality

Environment and Workspace
- Staging and Composition
- Viewing Zones
- Workspace Bounds
- Object Placement and Scale
- Viewer Scale
- Angle Awareness

4. Technical Guide

General Concepts
- Stereoscopic Display
- Angle Awareness
- Coordinate Systems
  - Camera Space
  - Viewport Space
  - World Space
  - Display Space
  - Tracker Space
- Head Tracking and Glasses
- Stylus
  - Buttons

- LED
- Vibration
- Tap
  - Enable Mouse Emulation
  - Enable Mouse Auto-Hide

System Architecture

Rendering Content
- Buffering
- Depth and Clipping Planes
- World and Object Scale
- Vertical Synchronization (VSync)
- General Performance Tips
  - Improving Responsiveness

Native Programming
- Software Development Kit
- Display Buffer Setup
  - OpenGL
  - Direct3D
- Stereoscopic Rendering

Unity 3D Programming
- zSpace Package and Getting Up and Running
- zSpace Plugin Architecture
  - StereoRig Game Object
  - Plugin Callback Events
  - Maintaining Camera Scripts and Effects
- Editor UI Properties and Debug Information
  - zCore Inspector
    - Debug
    - Stereo Rig
    - Glasses
    - Stylus
- Samples
  - Stylus Game Object Manipulation

System Setup and Diagnostics
- System and GPU Requirements
- System Check and Control Panel
- Demo Applications

5. zSpace Partners
Brand Guidelines
    Name Usage
    Color Palette
        Color Specifications
    Brand Marks
        Wordmark
        Portal
        Logo
            Horizontal Logo
            Vertical Logo
    Brand Mark Misuse
STEM User Interface
    Control Bar
    Tool Palette
    Backpack
    Inspector
    Context Menu
    Trash
    Help Screen
STEM Content Creation
    Studio Models
    STEM Activities

Chapter 1: Overview

Overview

zSpace creates a *mixed reality* computing experience that is immersive, interactive, and lifelike.

This document provides guidelines and know-how for designing and developing zSpace mixed reality applications. It provides guidance on how to think in 3D when creating experiences for the zSpace platform.

This guide is intended for a variety of audiences, including visual and interaction designers, developers, product managers, and program managers. This covers anyone who is responsible for designing new zSpace-enabled applications or porting existing applications.

For UX/UI Designers, this document describes best practices for recommended UX and UI design. It describes how to compose 3D experiences.

The Technical Guide section introduces general development concepts, native development, and Unity development.

For zSpace Partners, this document includes branding guidelines, references for our educational applications standard UI, and content authoring steps and requirements.

zSpace Basics

zSpace is a mixed reality hardware and software platform that enables developers and users to interact with computer-generated objects in a three-dimensional (3D) holographic-like environment, often likened to virtual reality (VR) or augmented reality (AR). However, the biggest difference with a mixed reality zSpace experience is that it lives comfortably on your desktop, and does not require the user to wear any electronics or heavy displays.

Observing objects and scenes in 3D is an extremely natural and efficient way to understand the complex spatial relationships of the world around us. zSpace presents computer-generated imagery in the same way that we see the world. This creates visual excitement and an instant understanding of complex structures that would otherwise be unfathomable if viewed on a traditional 2D screen.

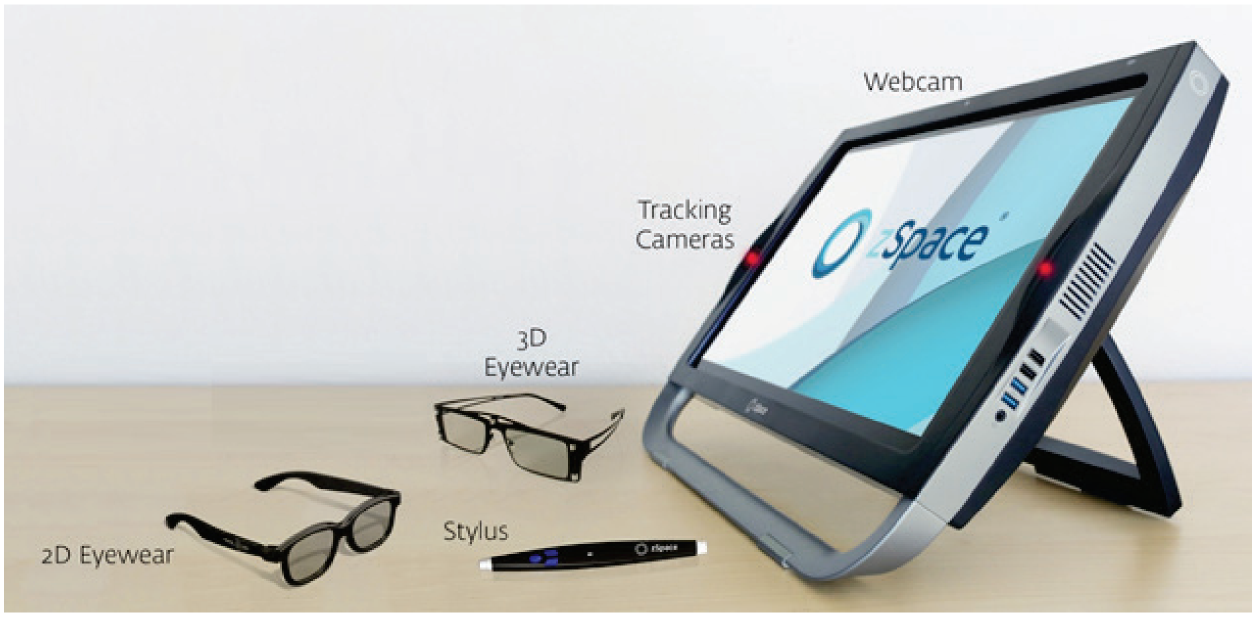
Some cognitive psychologists believe that interactive, stereoscopic 3D technology, such as the zSpace system, use the intuitive reasoning system, which is faster than our analytic reasoning system.

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

Standard Components of the zSpace System

zSpace hardware technology is currently available in two forms:

1. All-in-one Systems, combining zSpace technology with integrated CPU, GPU, and operating system.
2. Stand-alone Displays, including zSpace technology, to be connected to your own computer system.



Stereoscopic 3D Display

A high-definition (1080p HD) 3D/2D computer display, featuring an advanced mixed reality tracking system and angle-awareness sensor.

Eyewear

Lightweight, passive (no electronic parts) eyewear, required for the holographic 3D effect and head tracking.

Stylus

An active, pen-like input device, held in front of the display and used to reach into the 3D world. The stylus features six degree-of-freedom tracking, three buttons, haptic feedback, and an LED light.

Mouse

Standard 3-button mouse with scroll wheel.

Keyboard

A 78 key space-efficient keyboard. The keyboard includes function keys and supports configuring shortcuts.

Plus Bring Your Own Devices

You can attach peripherals to complement zSpace applications such as trackballs, inertial-sensor gamepads, and so on. Your application must be programmed to support such peripherals.

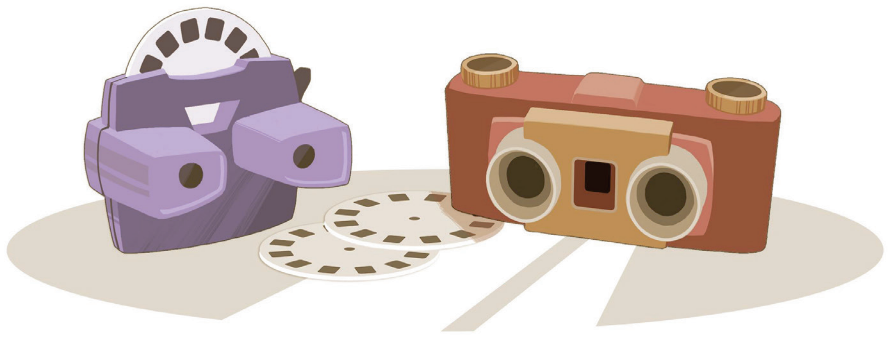
Chapter 2:
Mixed Reality Essentials

Mixed Reality Essentials

If you are new to the field of mixed reality, augmented reality (AR), or virtual reality (VR), this chapter helps you understand how the technology works. It describes some of the past techniques for simulating the real world viewing experience on a flat screen. It discusses the human vision system and how we need to accommodate it to create comfortable and engaging experiences. It also describes some of the current limitations of the system and how we can work within those constraints.

Introduction to Stereoscopic Imagery

Three-dimensional (3D) imaging has existed since about 1840 through the technique of stereoscopic photography. In this approach, a twin lensed camera—representing human eyes—captures the unique, offset left and right eye perspectives of a scene. When viewing the resulting stereoscopic pair of images, each eye receives its own left or right view and the brain recreates the illusion of space from that captured moment. The resulting 3D image makes the viewer feel as if they are part of the scene.



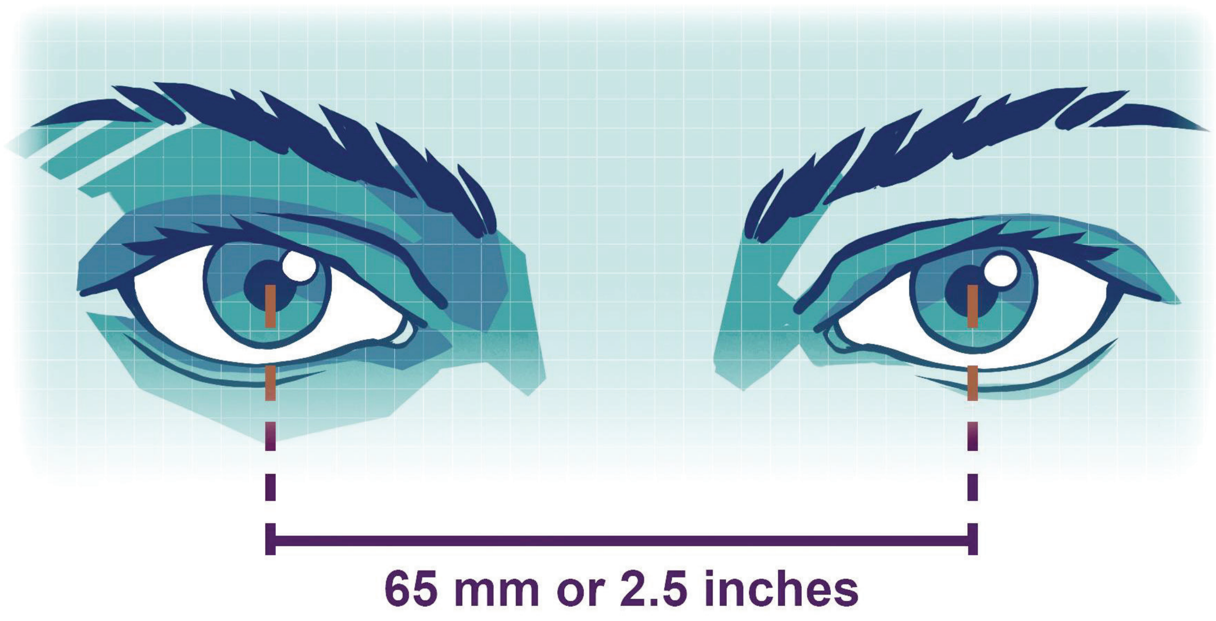
3D movies arrived as early as 1922 with “The Power of Love” and allowed the audience to experience this recreation of space, in motion, for the first time. Such movies have the same limitation as 3D imaging—the scenes are only visible from the predetermined viewpoint of the camera that captured images. The same limitation is seen in 3D movies today.

zSpace enables us not only to see 3D in motion but also allows us to look around the digitally-generated scene and interact with objects just like real life. This real-time 3D experience is a unique and natural way to view computer-generated imagery.

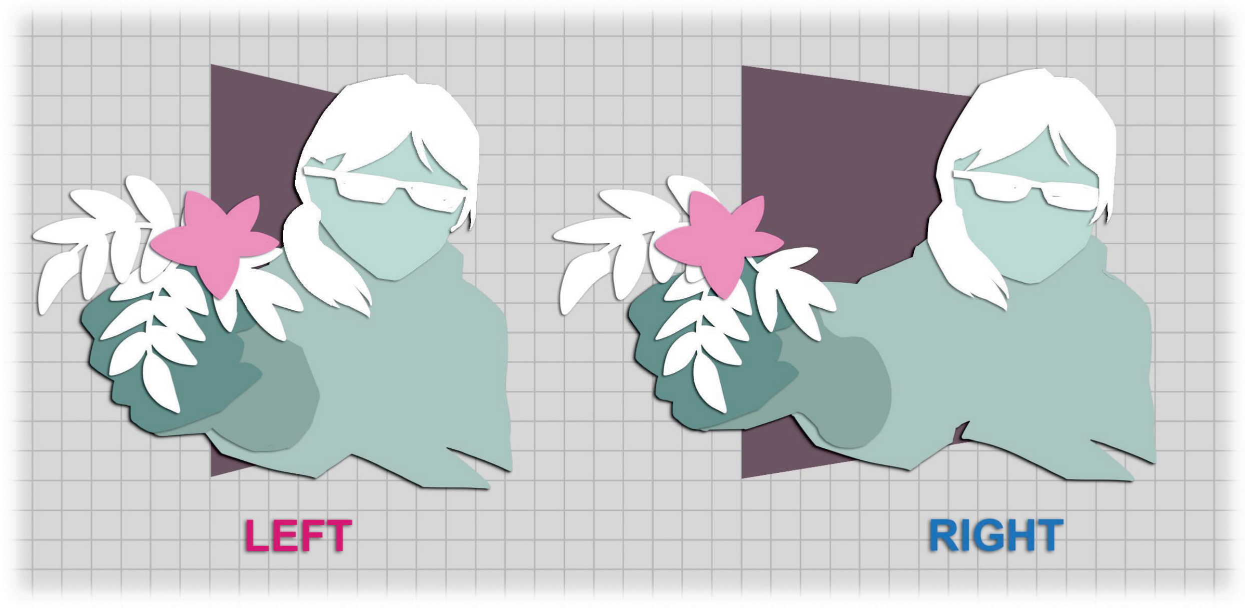
Human Vision

People have two eyes that point forward and we share this design with all hunting animals that need to judge distances with a high level of accuracy. The distance between our eyes is called the *interpupillary distance*. This distance can vary from about 45mm to 75mm, but averages about 65mm or 2.5 inches.

The ability to see out of both eyes simultaneously is called **binocular vision**.

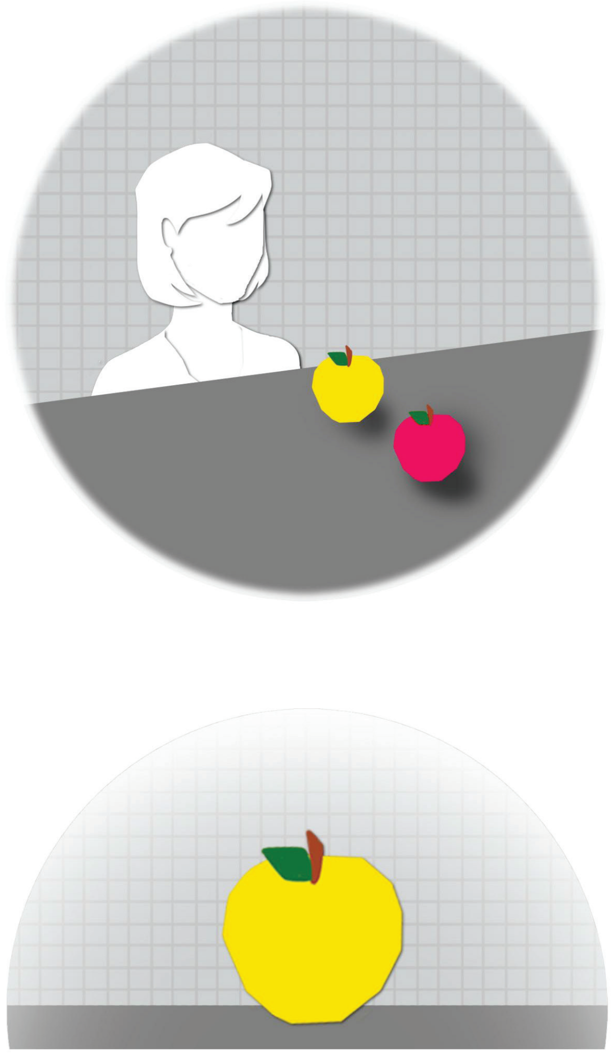


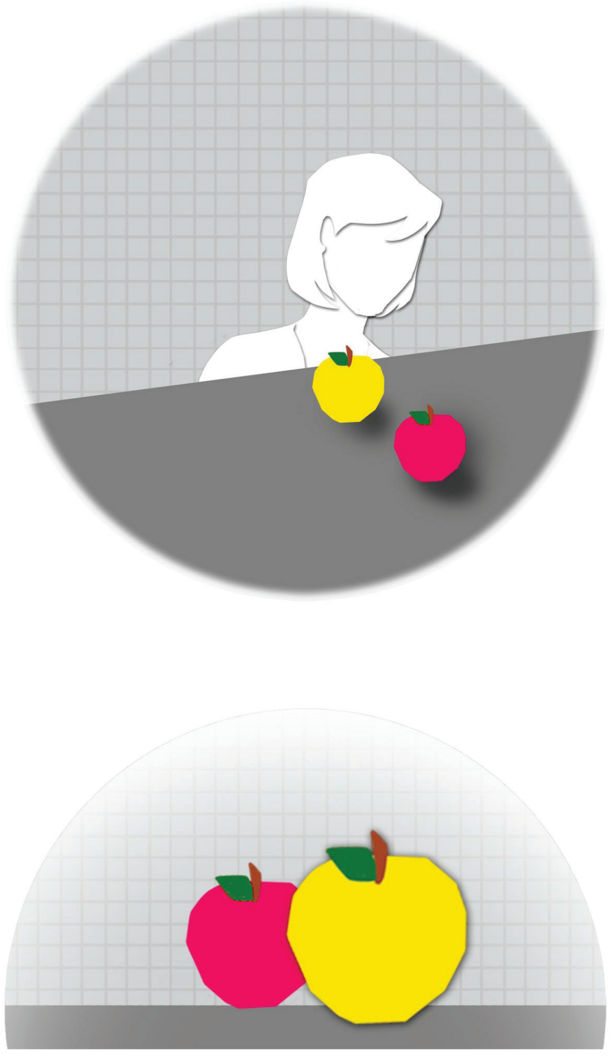
This separation allows our two eyes to see slightly different perspectives of the world. Our brain compares these left and right eye differences in order to understand the spatial complexities of the view in front of us.



Stereoscopic vision is the ability to perceive that single image in three dimensions.

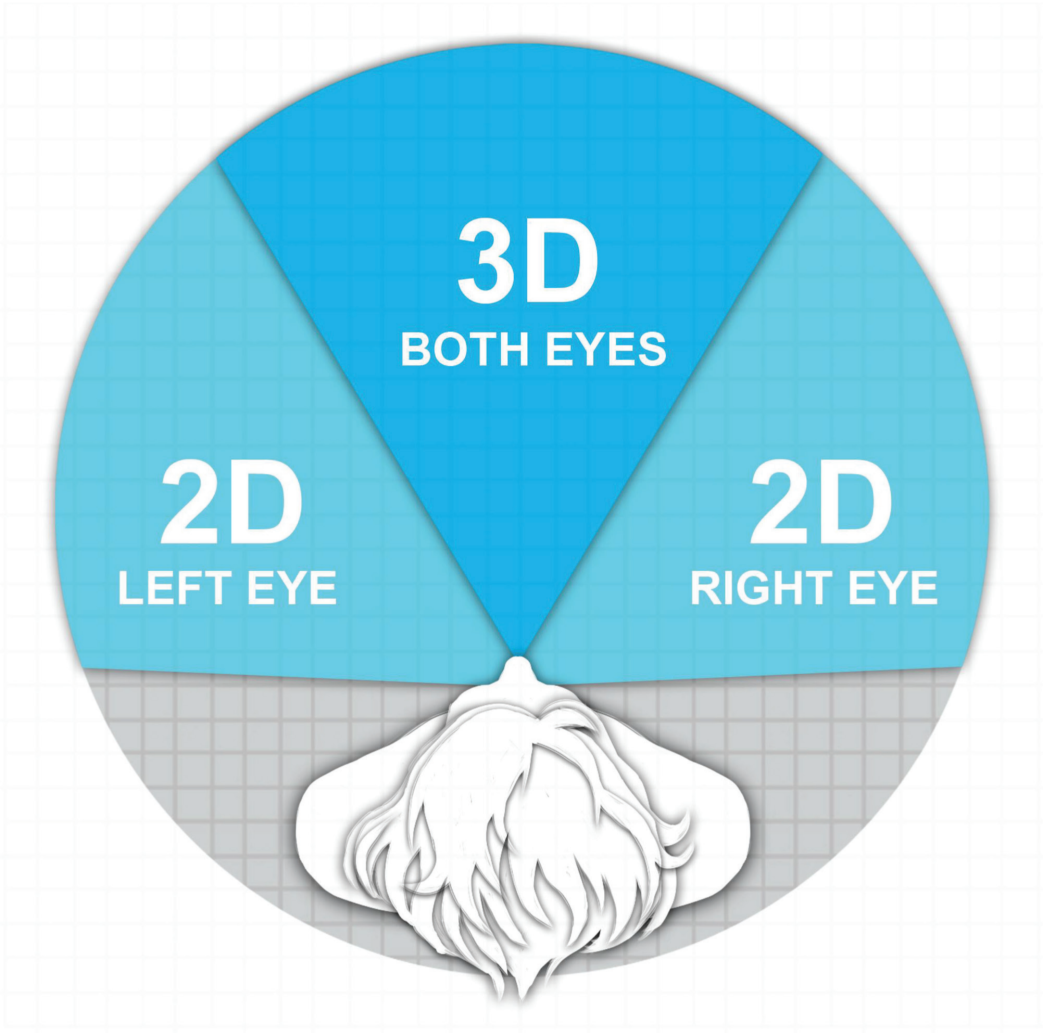
Another key aspect of stereoscopic vision, or vision in general, is that the view changes when you move your head. As an example, line up two objects on your desk, such as two coffee cups, one behind the other. Move your head to the left and right and notice that the back coffee cup appears and disappears 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.


EYE VIEW


EYE VIEW

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.

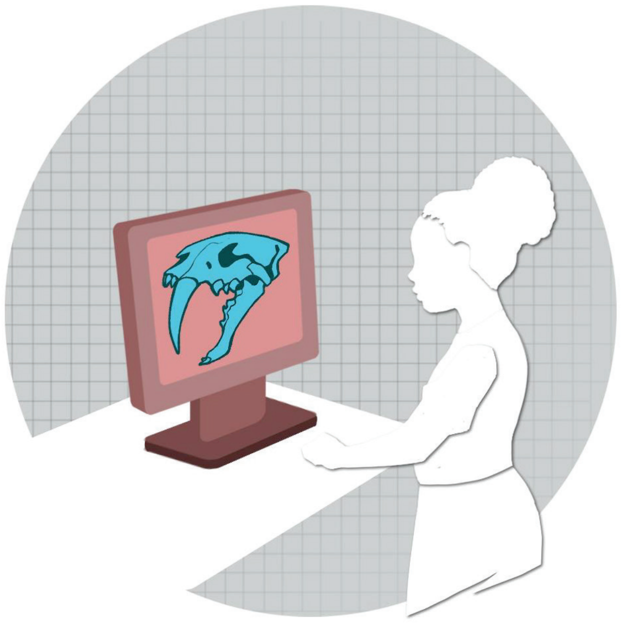
Although we can see far to the left and right, only the center of our vision system can see in 3D. The far left and right areas are mono as our nose partially blocks the view.

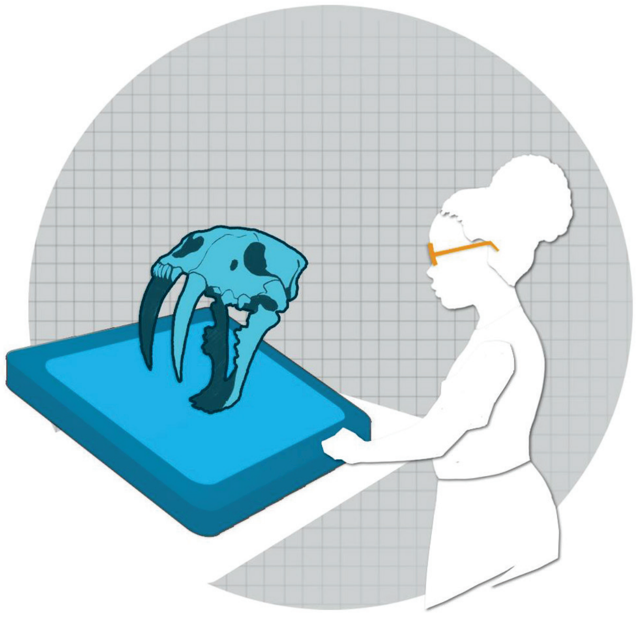


The 3D area of our vision system is a powerful tool for spatial understanding, and 3D zSpace imaging is a powerful way to understand the spatial complexities of computer-generated objects.

zSpace Vision

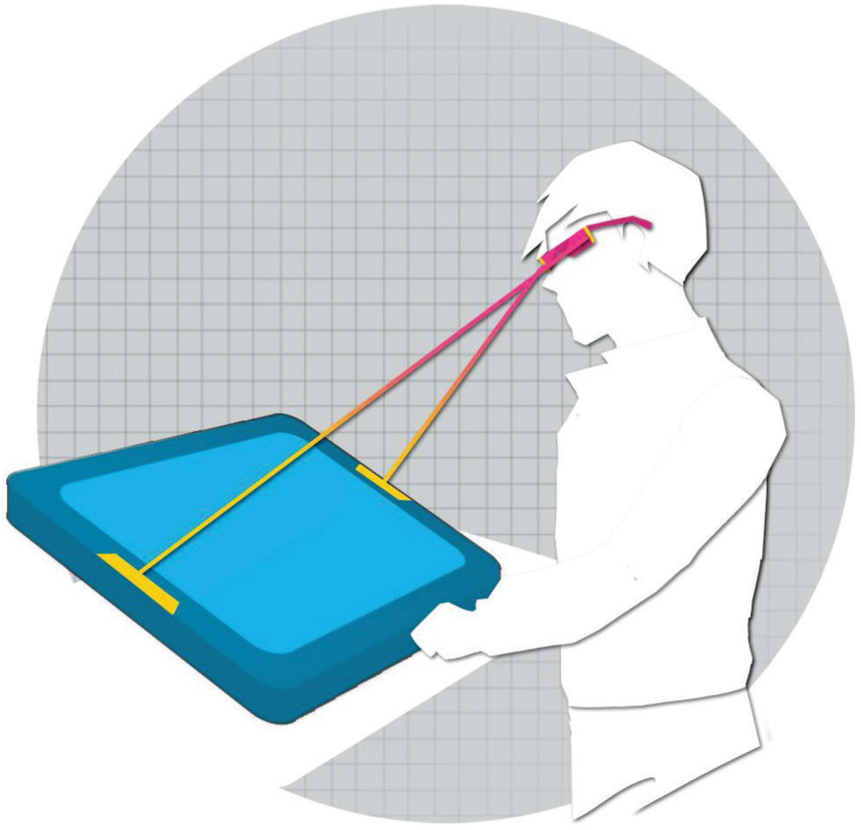
The zSpace platform uses computer graphics to generate the unique left and right views needed for stereoscopic vision. This happens in real time and makes it possible for any perspective of a scene to be generated instantaneously. This allows the user to look around the 3D objects by intuitively moving their body or by manual interaction using the zSpace stylus. The result is a sense of involvement with the natural, spatial realism of the recreated computer graphic scene. This spatial realism provides a vastly superior visual experience for understanding the structure of objects and scenes when compared to viewing traditional 2D computer graphics.



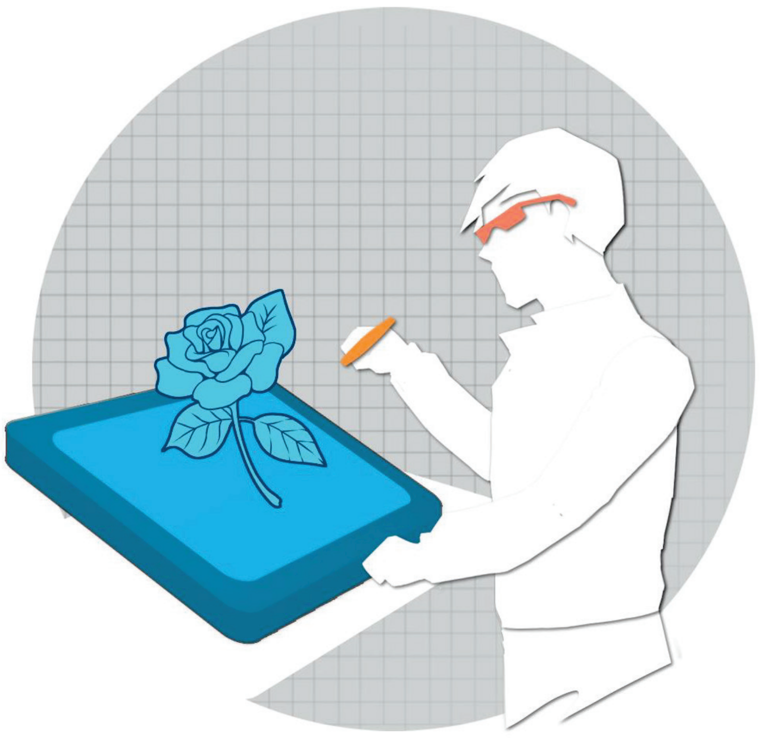


zSpace can create spatial realism because it determines the position of the viewer’s eyes relative to the screen surface. zSpace does this by following the tracking markers on the 3D glasses worn by the viewer. With this point of view information, zSpace accurately generates the correct stereoscopic perspectives on the screen surface to recreate the 3D scene from that specific angle of the viewer’s eyes. Every moment of viewing is a personal, custom-made, perspective view. The 3D glasses also perform the traditional 3D role of distributing the left and right stereoscopic views to the viewer’s eyes using circular polarization.

Note: 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.



If you observe someone else using zSpace, you can see the complex perspectives being generated in real time on the zSpace screen. Without the zSpace glasses, you see both the left and right eye images. Observing someone else's 3D perspective scene with your own zSpace glasses may cause eye strain because the scene is not drawn for your viewing perspective. You may experience less distortion if you are standing close to the viewer.



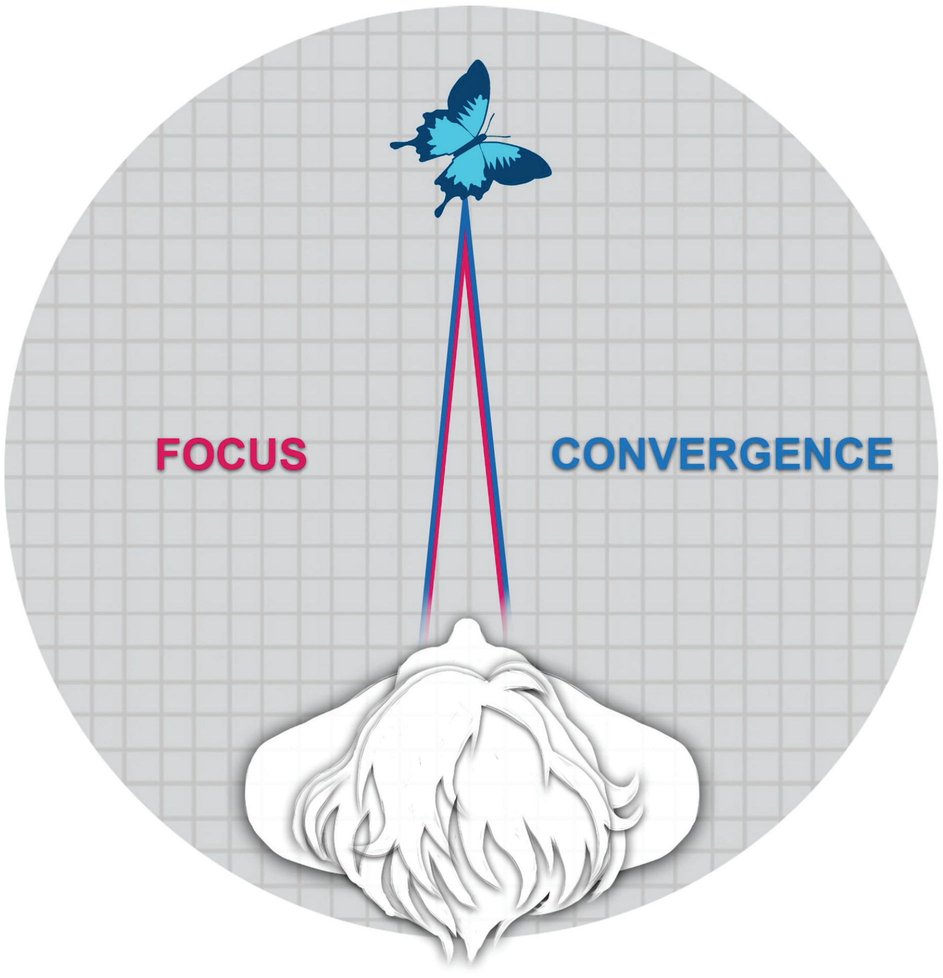
The same tracking technology is used to measure the exact position and orientation of the zSpace 3D stylus. This allows the user to “hold” and manipulate the computer-generated objects in space as if they were real.

Despite the complexity of generating the unique real time perspectives, the end result is similar to the first stereoscopic photographs: a left and right image, per a moment in time, displayed on the screen surface. The left and right images are displayed alternately at up to 60 frames per second (fps) per eye or 120fps combined. As each left or right image is displayed, the screen alternately polarizes the light so that the left image can only be seen through the left polarized lens of the glasses and the right image can only be seen through the right polarized lens of the glasses. The eyes relay the two unique images to the brain, as they do in real life. The brain puts the images back together, and the viewer experiences the illusion of depth as if it were real. See Chapter 4. Technical Guide for more information about the polarized filters used in zSpace stereoscopic display.

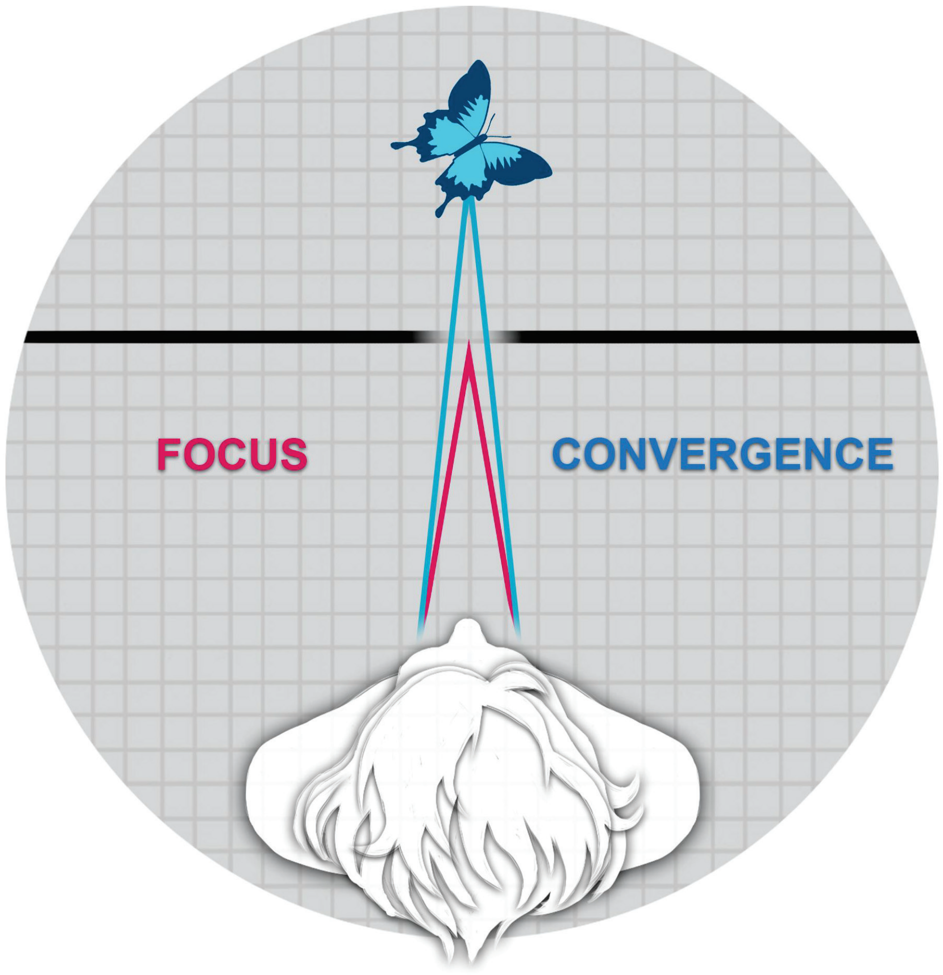
A stereoscopic photograph could be made to match the human viewing condition by choosing lenses with a field of view that matches the human eyes’ field of view, spacing the lenses apart to match the human eyes’ interpupillary distance and viewing the resulting 3D image life size. This is called an *orthographic* stereoscopic image.

Focus and Convergence

When we look at an object in the real world, both eyes converge on the object. As they do this, each eye's lens also focuses at the object distance (also known as accommodation). Converging and focusing become habitually coupled together at a young age.



When we look at the zSpace monitor, we must always remain focused on the screen surface in order to keep the image sharp, but we must converge at different depths to see the 3D illusion.



Convergence and focus would, therefore, appear to **de-couple** when viewing a zSpace 3D image, but this is not the case for normal scenes. *Decoupling* of convergence and focus does not occur if the stereoscopic depth remains within the human eyes' natural depth of field. Depth of field describes the additional distance that remains in sharp focus on either side of the actual point of focus.

Shallow depth of field vs. Deep depth of field

When a viewer looks at the bright zSpace monitor, an additional distance on either side of the screen surface remains sharp thanks to the human eyes' depth of field. Converging within this depth of field range does not require refocusing and remains **comfortable** for extended periods of time. Convergence and focus remain *coupled*.

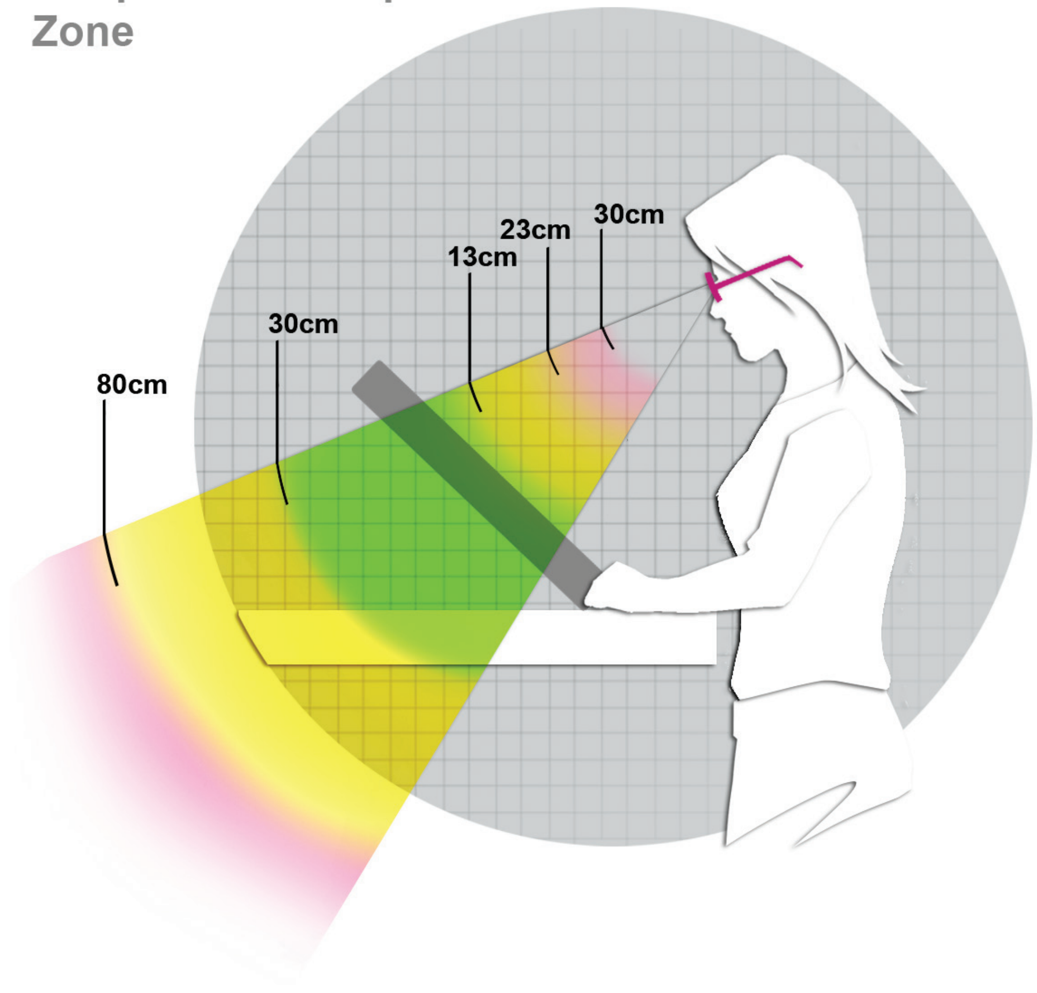
Designing for Comfort

This section describes some best practices to follow when designing an experience for zSpace.

Coupled and Decoupled Zones

The zSpace compositional space is like a stage made up of the coupled and decoupled zones, in front and behind the screen. Understanding the advantages and disadvantages of each zone helps you maximize the effect of your application.

Coupled + Decoupled Zone

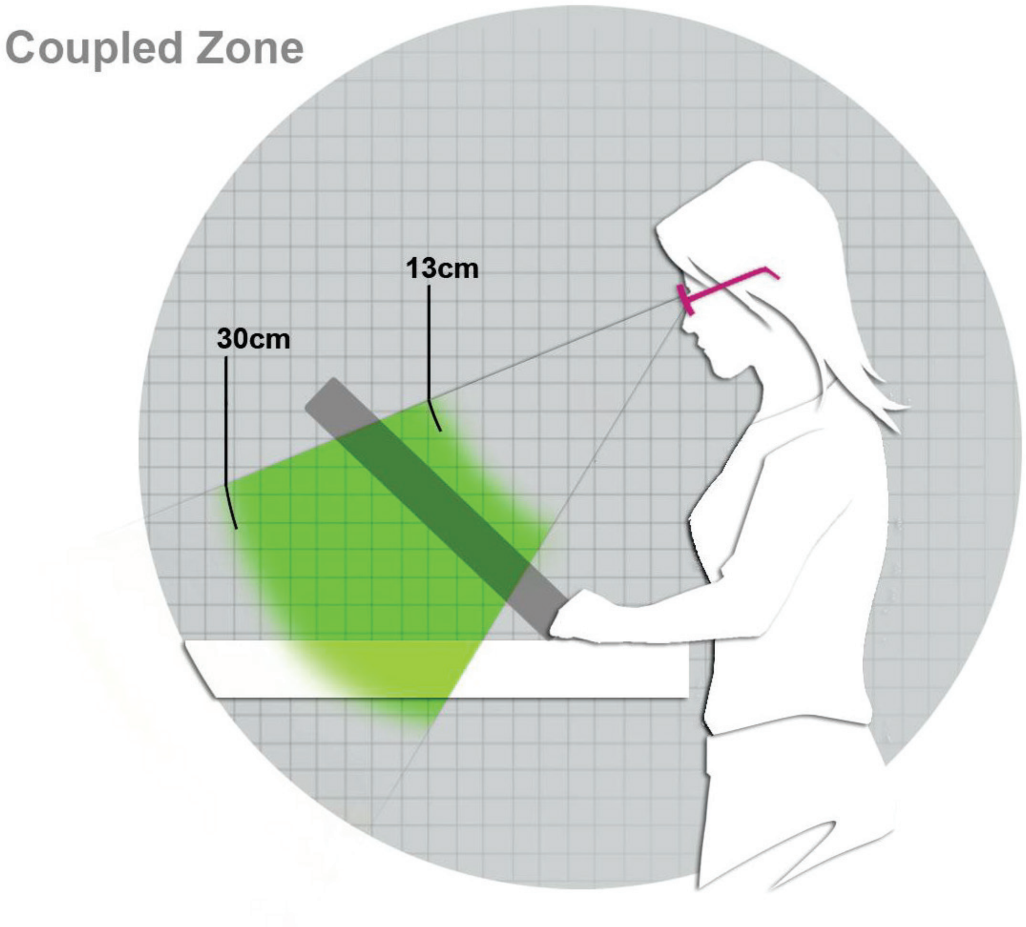


3D	Zone	In Front of Screen	Behind Screen
Comfortable	**Coupled**		
pixel		-100	+100
cm		13 cm	30 cm
inch		5”	16”
Strong	**Decoupled**		
pixel		-200	+150
cm		23 cm	80 cm
inch		9”	31”

Extreme	**Danger**		
pixel		-350	+250
cm		30 cm	∞
inch		12”	∞

Comfort Zone

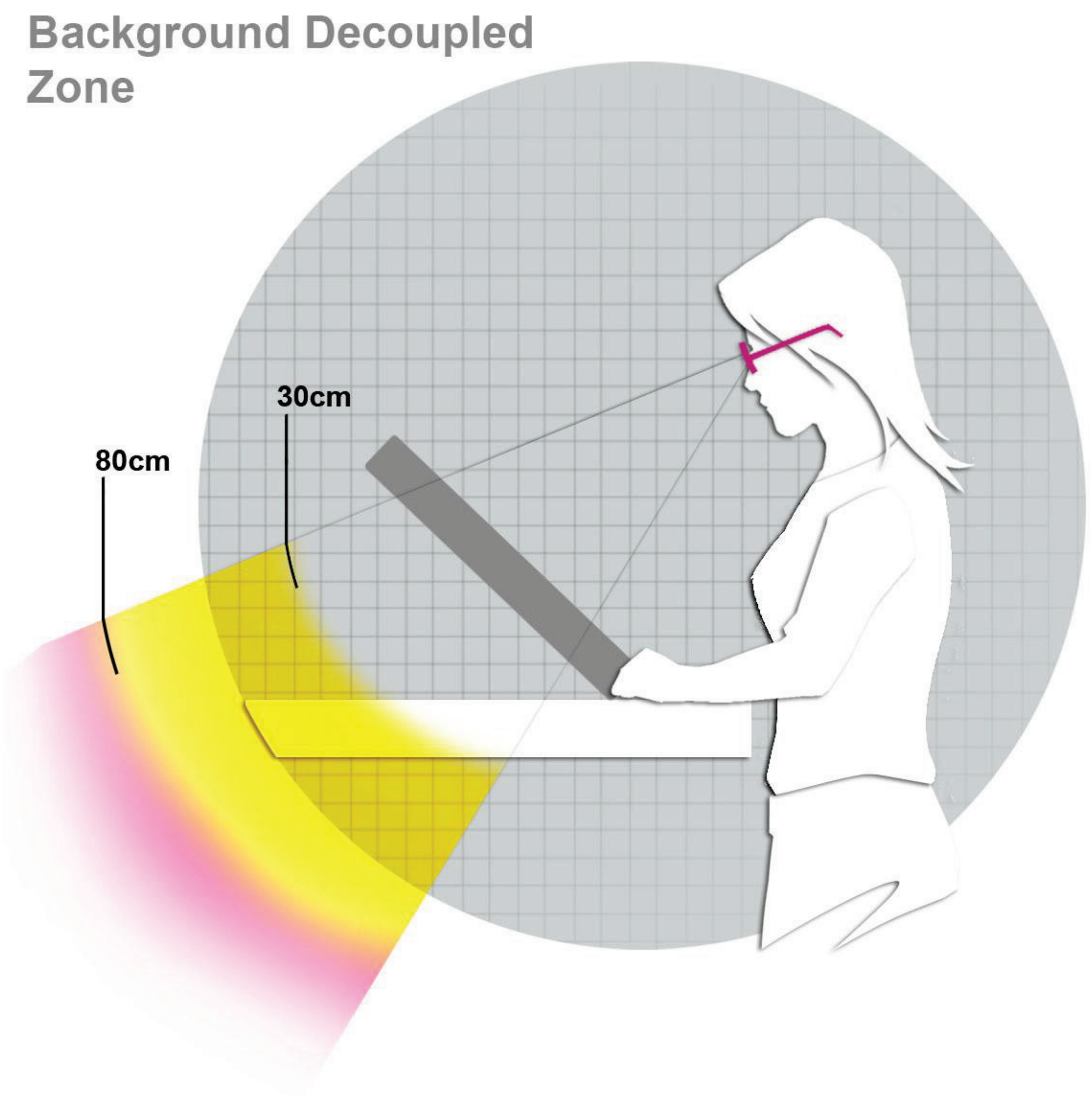
The *comfort zone* (also known as *coupled zone*) is the range of comfortable 3D viewing to use as the primary compositional space for most 3D scenes or applications on zSpace. The *comfort zone* is bisected by the screen surface. The closer an object is to the coupled zone, the more comfortable it is for users to view for long periods.



Creating 3D scenes with points of interest that go beyond the comfort zone is possible. Adding focal points outside the comfort zone should be limited and carefully controlled. The increased depth of field requires the viewer’s vision system to decouple convergence and focus. This can lead to eyestrain over time.

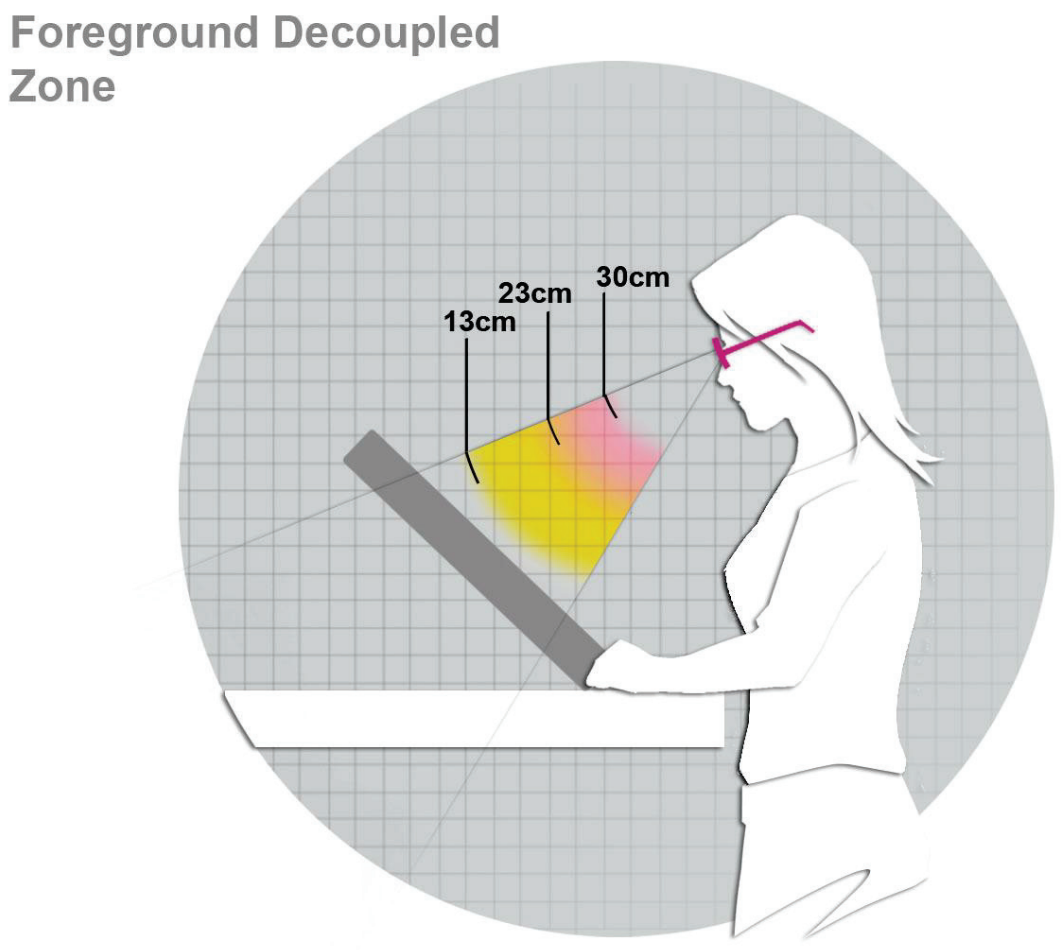
Background Decoupled Zone

The *background decoupled zone* can be used to extend the depth of the scene into the far background. Even if the viewer's attention remains inside the coupled zone, a deep background as peripheral depth can add a sense of vastness to any scene. Deep space or distant mountains can benefit from a decoupled background zone. Avoid high contrast areas if they display ghosting. Most people can view very deep backgrounds for short periods of time.



Foreground Decoupled Zone

The *foreground decoupled zone* is the most engaging of the zSpace zones. It allows objects to float out in space, disassociating themselves from the normal constraints of the screen surface. It is the “WOW” moment of any 3D experience. Many people associate 3D with this “pop out” effect and look forward to this powerful illusion when they view 3D. Applications should deliver this effect either through shock and surprise or by encouraging users to pull objects towards them, in front of the screen, under their own control. Either way, durations of pop out 3D should be kept relatively short to avoid eyestrain.



Displaying Objects in the Decoupled Zone

The *decoupled zone* is anything beyond the comfort zone depicted in the previous figure. As the decoupling increases, the 3D effect increases but so does eyestrain. At a certain point the 3D illusion breaks completely as the viewer is unable to fuse the two images of the stereoscopic pair. This limit is a little different for each person.

The decoupled zone can be used for bold 3D effects of short duration. You can also use this zone if the decoupled depth only contains peripheral objects that fill the scene but do not require focused attention. In this case, the viewer should remain converged within the coupled zone while appreciating the extreme depth in their peripheral vision.

Direct viewing of decoupled objects is much less problematic if the viewer is in control of the depth. zSpace allows the viewer complete control of an object. For example, the viewer can move an object into an extreme close up position that is far outside the coupled zone. On zSpace this is not an issue as the viewer can choose their own level of acceptable decoupling as they examine an object. If the viewer feels any discomfort, they naturally reduce the depth by returning the object to the coupled zone.

Extreme 3D effects are impossible to create with a 2D monitor and are one of the most striking aspects of zSpace. The viewer should always be in control when viewing 3D beyond the coupled zone unless the aim is to intentionally create a visual shock. If the viewer does not control the depth of an application, then the comfort zone should be the primary target and extreme 3D can be used sparingly as needed.

Terminology: The foreground decoupled zone is sometimes referred to as the *negative parallax* zone, while the background decoupled zone is sometimes referred to as the *positive parallax zone*. In this usage, *zero parallax* refers to objects appearing exactly at the surface of the display, where convergence and focus match.

Window Violations

Window violations are most disturbing when a foreground object overlaps the vertical border of the monitor bezel. The border of any stereoscopic image creates a virtual window in space. Objects can appear “behind the window” or out in front, “through the window” towards the viewer. The stereoscopic window of zSpace is formed by the black bezel of the monitor and sits at the screen surface.

Objects behind the bezel or window are almost always comfortable to view. Objects in front of the window can also be comfortable as long as they stay in the center of the screen and avoid hitting the window edges. Once an object closer than the screen hits the window edge, it creates a disturbing visual effect. Our brain struggles to resolve how a foreground object can be cropped by a window that is further away. An image of a person’s head would look like it has been cut in half by the window.

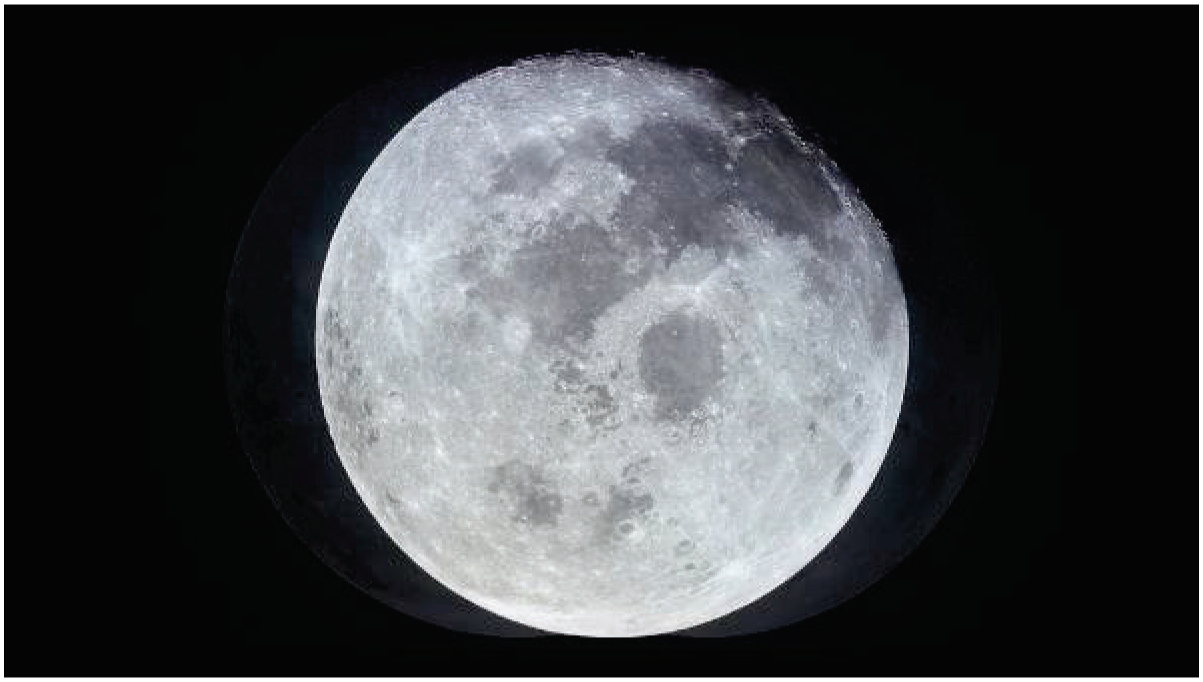
Top and bottom window violations are minor because our eyes are aligned horizontally and only see stereoscopic data from verticals. Left and right edge violations can be very disturbing and should generally be avoided.

Luckily the zSpace system easily allows the user to fix any momentary window violations by simply moving the object away from the edge or by physically shifting their point of view.

Disturbances caused by window violations are dramatically reduced in a moving scene. Objects can comfortably float out in front of the screen, cross the stereoscopic window and disappear without creating a problem. Only when the object stops on the edge, half masked and half visible, does it become a real distraction that could lead to eyestrain.

Ghosting

Sometimes the image meant for one eye receives some unwanted light from the image meant for the other eye, creating a faint ghost image, an effect known as *ghosting*. Ghosting occurs because glasses-based 3D systems cannot completely separate the left and right image of a stereoscopic pair. Ghosting is most visible in high contrast, highly separated images. A bright moon in the far background over black sky can create ghosting.



If the high contrast object is positioned at the screen surface, ghosting is eliminated, but this may also eliminate the 3D effect. Reducing the overall depth of the scene does little to help ghosting and should be avoided as a solution. Removing the 3D is the last resort.

The best way to avoid ghosting is through careful design and art direction. Reducing the contrast of problematic objects, softening edges, and increasing the surrounding texture all help to minimize the negative effects of ghosting.

To eliminate ghost images, follow these guidelines:

- Avoid using strongly contrasting colors, hues, values, and patterns.
- Do not use pure black or pure white for objects or backgrounds.
- 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 appear if it is in high contrast.

Sometimes you cannot modify an object 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 need 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 also eliminate flickering.

Depth Changes

When 3D scenes change rapidly in 3D movies at the cinema, the audience's eyes have to jump from one depth to the next in quick succession. We rarely do this in real life and it becomes exhausting over time. zSpace applications typically avoid this problem as the nature of content allows for long viewing durations between depth changes. If rapid changes of objects or scenes are required, it is important to match the depth from one scene to the next so that the viewer's eyes do not have to jump.

Depth Cue Consistency

The human visual system integrates multiple depth cues to perceive the relative depth of objects. Focus and convergence are an important pair of depth cues previously discussed in this chapter.

Convergence and *occlusion* (occultation*)* are another important pair of depth cues. Consider two objects, one in front of the other, so that the front object is closer to the observer and partially occluding the back object. Convergence depth cues should match, meaning not only should convergence for the first object be at a sharper angle, but the first object should not in any way be occluded by the second object. If the second object occludes the first, then the convergence cue (suggesting the first object is nearer) conflicts with the occlusion cue (suggesting the second object is nearer).

Convergence-occlusion depth cue conflict often occurs if a 2D user interfaces is overlaid above a 3D scene. See Chapter 3. Best Practices for a deeper discussion.

Performance Requirements

For quality and comfort, applications targeting zSpace are strongly advised to operate at no less than 45 frames per second, and recommended to generally operate at 60 frames per second.

In comparison with other forms of VR, such as head-mounted displays (HMDs), latency and framerate demands are less critical with zSpace, which is able to maintain a physiologically comfortable and immersive experience in a wider range of system conditions. Nevertheless, for the best experience, applications should be as responsive (low latency) and fluid (high frame rate) as possible.

Chapter 3:
Best Practices

Best Practices

This chapter describes best practices for user interaction, input devices, zView augmented reality, and the mixed reality environment.

User Interaction

A mixed reality environment is richer and more complex than a traditional 2D computer display. Design with 3D space in mind. Your zSpace application is more engaging if it builds on user experience and knowledge. Objects in mixed reality should mimic the real world, but this need not limit creative or magical experiences.

Everyone grows up learning how to handle, hold, and manipulate basic objects in the real world; moreover, most people know how to use computers, laptops, tablets, and smartphones. When implementing a zSpace application, take advantage of all of this acquired knowledge—both real and digital—to make mixed reality easier and more intuitive.

zSpace lets people directly manipulate objects in stereoscopic 3D in a volume in front of the screen. For example, you can rotate, reposition, resize, or zoom in on an object. Using zSpace, computer interaction becomes intuitive because we already know how to move objects around in the 3D world.

Stylus Behavior

Direct Interaction

In the real world, people are familiar with pointing at objects and grabbing them. The zSpace stylus is a natural extension of pointing at objects using a pen, pencil, or laser pointer. People quickly build on that experience by grabbing things in a 2D/3D environment; here, using a physical button press on the stylus is similar to grabbing with thumb and fingers.

When the user presses the center button and pulls back on the stylus, the object appears in 3D space (negative parallax) between the user and the screen. The stylus can provide the application with X, Y, and Z coordinates about its location, as well as its angle (yaw, pitch, and roll), known as 6 degrees of freedom (6DOF).

Depending upon your application design, the user can further manipulate the object by dissecting or taking apart the object, or performing other application-specific operations directly on objects in the virtual workspace.

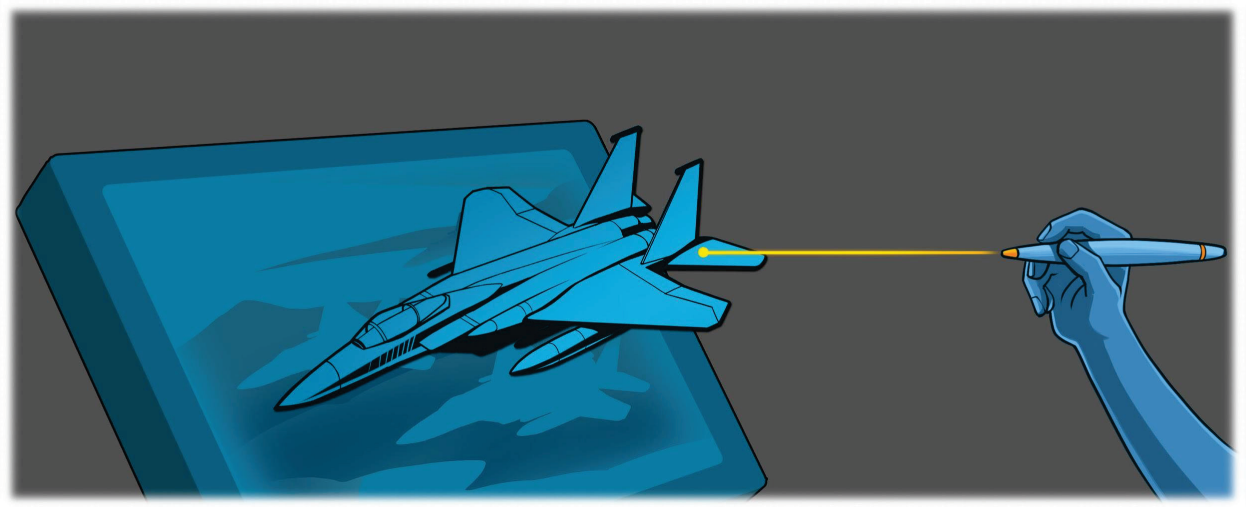
While unconstrained direct interaction is often most intuitive, sometimes you should add constraints to make it easier for the user to complete specific tasks. In some cases, you may constrain input to fewer degrees of freedom, such as limiting input to the x and y axes, or held objects to a specific plane or surface in the scene. Another constraint could be to snap-to-grid or snap-to-orientations (up, 90 degrees, and so on). You can let the user enable and disable constraints, or decide whether specific tasks should always assist the user with constraints.

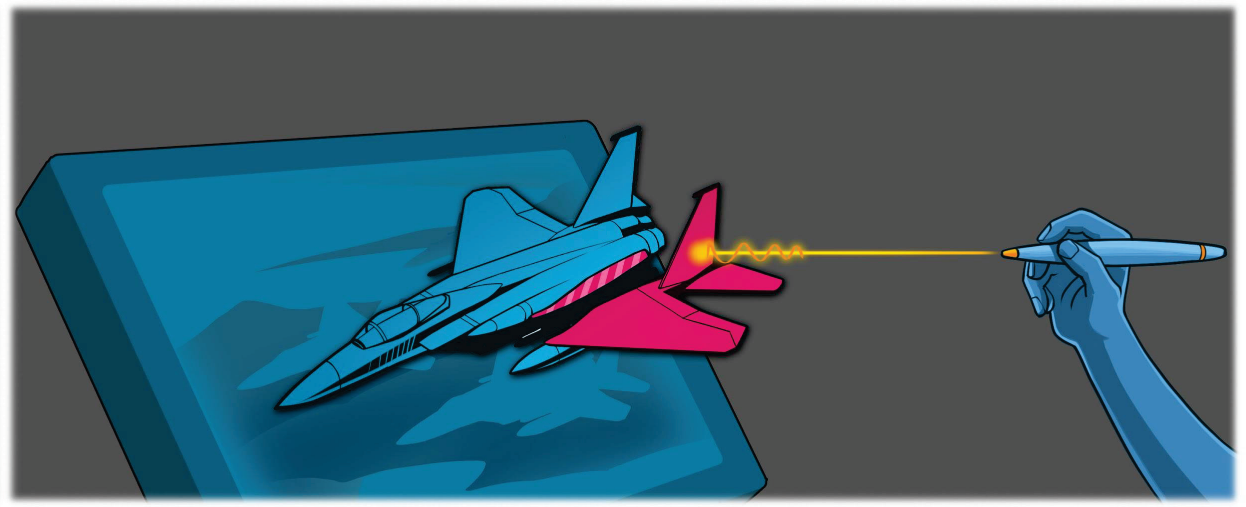
Visualization and Length

Just as a mouse controls an on-screen cursor, the stylus is often easier for a user to visualize when combined with an in-scene extension. This extension can take different forms, such as a laser pointer “beam”, a hook, or a hand. The stylus can even take the form of a 3D tool corresponding to the current software mode such as a scalpel, pencil, or screwdriver.

Beam visualization:

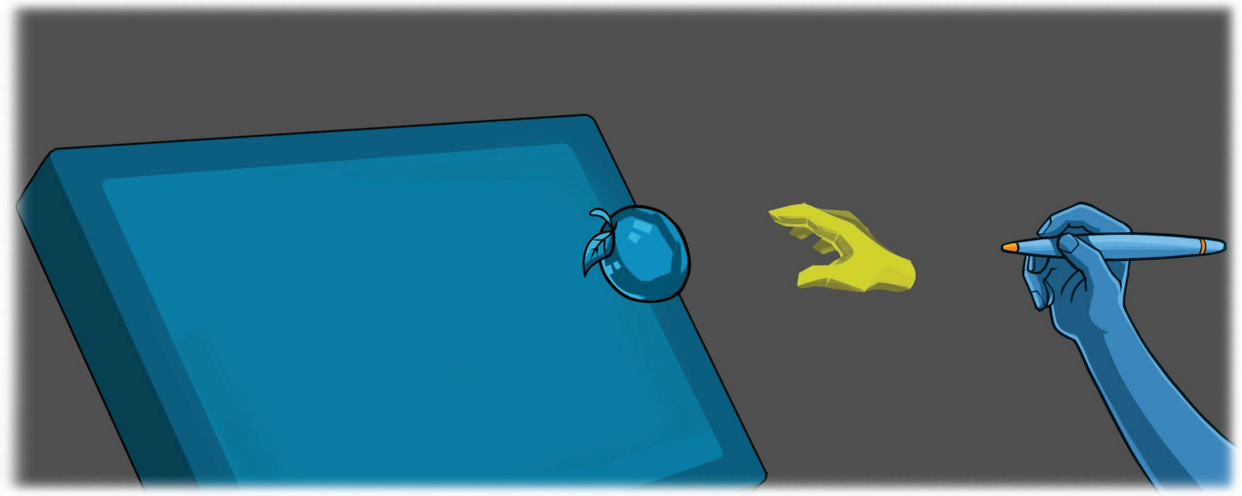
- A beam extends from the physical stylus along the stylus’ orientation.
- A tip (such as a cone shape) appears at the end of the beam to indicate a clear depth and location of the reach of the virtual stylus into the scene.
- The tip or beam may also change in color or shape according to contextual capabilities or tool modes.

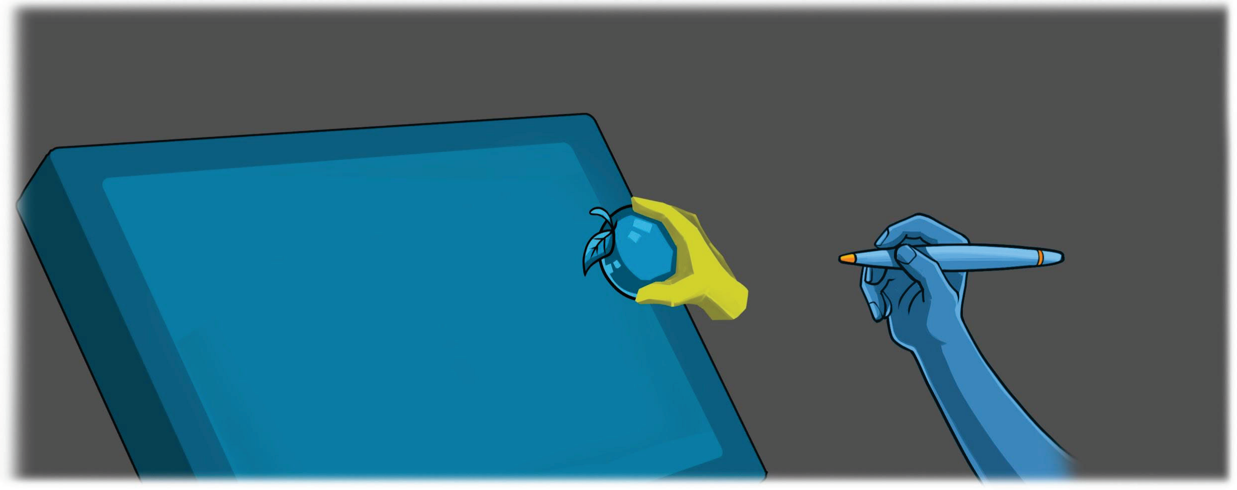




Hand visualization:

- A floating hand appears in-scene, positioned some distance away from the physical stylus, along the direction the physical stylus is pointing.
- The hand may change in shape, gesture (such as fist or open palm), color, and so on according to contextual capabilities or tool modes.





Tool visualization:

- A scalpel or knife can give the user a visual cue that encourages them to dissect or cut.
- A pen or pencil adds to the experience of writing in-scene.
- A measuring device can be used when appropriate such as a ruler or compass.
- A screwdriver or wrench can prompt users to assemble or take apart something.

The tool may change in shape, color, and so on according to contextual capabilities or tool modes. A beam or tool visualization generally feels more precise than a hand, while a hand can feel more fun. Choosing a visualization for the stylus' representation in the scene can have a large effect on the feel of your application.

Whether using a beam, hand, or other visualization, there are at least three approaches to pointer length or offset:

- **Fixed length**: A fixed beam's length remains constant as it moves through the scene. 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, or in scenes which contain many layers of partially transparent information or objects. 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, your application can use volumetric selection. Volumetric selection lets the user click & drag to select multiple objects in a 3D area.
- **Variable length**: 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 the adaptive beam for new users or to provide a short learning curve.
- **Hybrid (or adaptive) length:** The beam's length varies as with variable length, but has a maximum length, apparent when there are no objects in the scene or when the stylus position and trajectory is far enough away that it does not touch any objects.

Most zSpace applications use hybrid or variable length stylus beam. You can include each mode: some tools work best with one type. For example, a 3D spline-building tool requires a fixed-length beam because the user must place points at specific locations in empty space. The selection tool can work with either a variable or fixed-length beam. You can also let the user switch between the fixed beam and variable beam.

If using a cursor visualization which extends continuously from the physical stylus, it is important to maintain alignment between the physical stylus and the visualization so that they appear connected. 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.

Rotation

While there are benefits of direction interaction, some operations can be as awkward here as in real-life. Rotating the held stylus is of course limited by the physiological constraints of the human wrist. Users often want to see the back, top, bottom, or sides of an object. To do so purely with direction interaction requires the user to repeatedly grab, rotate, and release.

If users of your application want to view all sides of an object, consider helping them by scaling the rotation effects of rotating the stylus. You can think of this like power steering in a car. For example, once a user has grabbed an object, if they twist their wrist (twist the stylus) 20 degrees, then scale this rotation so that 30 degrees is applied to the object.

Other approaches to aiding with rotation include complementing the stylus with other devices such as keyboard or trackball, discussed later in this chapter.

Buttons

The zSpace stylus features three buttons, like most mice. However, the stylus buttons are arranged in a different pattern than the mouse, which can be beneficial but requires care in approach and user training.

Many zSpace applications adopt this standard mapping:

- Center button (primary button): Select objects and activates UI controls.
- Right button (secondary button): Open context menus.
- Left button (tertiary button): Activates application-specific functions.

With the exception of center and left buttons swapped, these mappings resemble the default mouse behavior, which provides some familiarity to mouse users.

NOTE: All functions 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 easier access methods for novice users.

If your application does not use all three buttons, decide how to handle the user pressing a non-functional or unmapped button. If nothing happens, users might think the stylus or application 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, or a fun Easter egg, if the user presses a non-functional button.

Selection

In addition to direction interaction with objects, you can use a selection state approach. This is useful for enabling multi-selection. Multi-selection lets the user deal with multiple objects and avoids requiring tedious actions, such as dragging objects to a trash can one-by-one.

For example, you might associate the left stylus button, or center stylus button plus a keyboard modified key, to add and remove objects to a selection group. Visual feedback is used to easily identify which objects are part of a group and which are not. Once objects are in a group, the user can manipulate the group as a single object. The user can move the group of objects to a new position in the scene, remove the group of objects from a scene all at once, or resize all the grouped objects as an ad hoc unit, and so on.

Context Menus

Context menus can be as useful in 3D as in 2D. The challenge with context menus in 3D is placement. In 2D, a context menu can always comfortably be drawn in front of any other context.

In 3D however, the context menu itself needs to have consistent depth cues, usually meaning the context menu must exist as an object in the 3D space. In turn, the challenge becomes finding appropriate placement for the context menu. Some options include:

- Position the context menu around the stylus beam, near the object.
- Position the context menu at the surface of the object being interacted with.
- Position the context menu at one of a set of fixed locations, temporarily pushing content away from the user if needed.

See Chapter 2: Mixed Reality Essentials for more information about depth cue consistency.

Text and Legibility

For applications which depend on presenting significant amounts of text, legibility is very important for comfort.

Text is most legible when at zero parallax (i.e. appearing exactly at the surface of the display). This is because ghosting is completely absent here, and because it is an opportunity to benefit from sub-pixel font rasterization (depending on your choice of UI framework). If your overall design would place text near zero parallax, consider adjusting so that your text is exactly at zero parallax.

While in general, high contrast text is easier to read, you must watch out for too much contrast if displaying your text too far into negative or positive parallax, where ghosting becomes more apparent. If you must display your text far from zero parallax, choose a medium brightness text color over a light grey or off-white background.

Audio

Designing an experience where objects and UI each have unique sound, especially in combination with haptics, can heighten the magic and quality of the experience. The greatest potential here is with in-scene objects and spatial audio.

For example, imagine a scene featuring a fully animated and scripted parrot. The user can hold, feed, and move the parrot. You can elevate the realism of the experience using a variety of sounds, such as squawking, crunching seeds, flapping wings, and so on.

Wherever the object is in the scene should be where the sound appears to emanate from. Especially engaging are subtle sounds which can only be heard when the object is brought closer to the observer, such as the foreground decoupled zone. Implementing spatial audio correctly requires knowledge of whether the user is using headphones or speakers, and in the former, is dependent on the location of the user's head, provided by the zSpace tracking system.

Haptics

The stylus includes a vibration motor which can provide haptic feedback to the user. The haptics can be controlled using software APIs. You can vary the strength, duration, and pattern of vibration. In general, haptics can be used to improve user experience in these ways:

- Simulate real world effects such as feeling a beating heart, recoil of a gun, collision of objects.
- Indicate success or failure of an action, such as selecting a virtual object, or snapping it into place.
- Indicate intersection of the stylus beam with selectable object, UI element, or icon.

Because haptic feedback is intimately associated with the user's hand, it becomes a visceral part of the user experience. Consider the user reaction and test haptic feedback before implementing it. Use haptic feedback sparingly, so that the user isn't constantly feeling the stylus buzzing. Engage haptics only when relevant events happen. Sparing and selective use adds realism into the scene and a delightful (sometimes surprising) user experience.

Visual Feedback

As the user interacts with objects that represent the real world, you can provide feedback that matches real world behavior. 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.

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. Generally, users must receive immediate feedback about physical actions (button pushes, object selection) within a fraction of a second or they believe the action failed and try again.

For the in-scene objects, 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.

For long running actions taking 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 matches the user's expectations based on experience with 2D user interfaces.

Permanence and Magic

An important concept humans learn in infancy is *permanence*, or the understanding that objects (generally) don't come and go in and out of existence. We learn early on that objects actually just move in and out of view, or at a later age, that objects combine or break apart into new objects.

Especially in AR/VR, permanence is important for increasing the realism and immersion of the experience. For example, deleting an object by dragging it into a trash can has a much more tangible and believable feeling to it than deleting an object and seeing it instantly disappear.

Where permanence becomes infeasible is where *magic* steps in. Humans also learn to believe (through the arts and media) in magic, or more accurately to say, most people have a learned capacity for suspension of disbelief regarding magic. The word *magic* can apply to more than spells and the four elements; magic can be specialized objects which magically come in and out of view where justification is needed for suspension of disbelief.

For example, suppose your application has a *clone object* operation. Rather than simply producing a cloned object instantly when activated, you might animate (from off-screen, for permanence) in to the scene a scanning platform under the target object and play a scanning animation effect. Then animate the platform to open space and spawn and animate the cloned object up from the platform. Finally, animate the cloning platform off-screen. The technological magic in this example works because it does not break permanence and immersion.

Broadened Demographics

Handedness

The hand with which the user holds the stylus can have implications for your UI design. Reaching into the scene with the stylus using your right hand can occlude the visibility of UI and objects in the lower-right region of the screen. Similarly, your left hand can occlude the visibility of UI and objects in the lower-left region of the screen. Moreover, the further the user must reach into the scene to access a UI element or object, the more effort must be exerted. By being aware of user-handedness (inferred from the stylus pose, or known by user setting), your UI and object placement can adapt to user handedness. For example, if you have a Frequently Used Tools toolbox, it can change position to be most convenient for the handedness of the user.

Motor Skills

Users of different age and agility have different degrees of fine motor control. Users that are especially young or old may have difficulty selecting or activating small or moving objects. Users generally tend to affect their stylus trajectory unknowingly when pressing the physical stylus buttons, dipping down, and sometimes leading to false negatives in your UI.

Object and UI sizes previously optimal for mouse pointer or touch-screen may not be optimal for stylus use. Larger hit areas can help these users, as well as activation logic dependent on the depression of the stylus buttons rather than the release.

Color Blindness

While comprehensive design for color blind users is beyond the scope of this guide, some general tips include using changes in brightness rather than hue to indicate state changes, such as hover effects, object selection, and so on. This could be dark green to bright green, or dark blue to light blue. Avoid switching between two dark colors or two light colors.

Features unique to stereoscopic 3D can be used to your advantage for color blind users. Motion parallax and binocular parallax inherent to zSpace may also help color blind users differentiate between objects or boundaries that would otherwise be challenging to distinguish.

Undo-ability

Make sure your application encourages exploration by offering mechanisms to undo actions, redo actions, return to prior saved settings, and return to default settings. This is consistent with user expectations of 2D applications. For example, many users have memorized the keyboard shortcut for Undo (**CTRL+Z**).

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 by displaying warning messages. For example, if a user is about to exit without saving changes, something which may not be reversible, your application should prompt the user to verify their intention.

2D User Interfaces

Using a traditional 2D user interface is generally easier when the UI is *screen-aligned* at zero parallax rather than arbitrarily placed in the 3D workspace. Keeping the 2D UI screen-aligned also creates a distinction between what the user perceives to be within or not within the 3D workspace. Design implications may include ignoring physics and collisions between the workspace and screen-aligned UI, though some fun behaviors can be created by allowing the opposite, termed “breaking the fourth wall.”

Like a HUD in a computer game, the screen-aligned UI may either be permanently displayed or temporarily displayed as needed. Reasons for using screen-aligned UI for traditional UI include:

- Text is easier to read in 2D.
- Readable by multiple users at a time, without the need for zSpace eyewear.
- More natural support for mouse interaction, which traditional UI is designed for.

Be sure to also provide consistency and predictability. If a menu item is not available at a particular time, change it to appear disabled (grayed out) rather than completely hidden.

Note: A 2D UI does not need to be flat. This may sound contradictory. For example, icons and buttons can feature 3D shapes or thickness, even when aligned with the screen. You can achieve the best of both worlds—providing the usability of a traditional UI with a 3D aesthetic.

It is important to prevent a 2D UI from interfering with your in-scene objects, which can create conflicting depth cues (see Chapter 2. Mixed Reality Essentials). Moreover, objects in the scene should never prevent the user from interacting with the 2D UI. To help meet both conditions, consider these possible solutions:

- **Temporarily make the in-scene objects translucent** – When displaying a control, gradually fade any interfering objects to become translucent and non-interactive. The gradual change prevents a blinking effect as the stylus moves around the scene. This approach maximizes available 3D space.
- **Reserve space for your application-level controls** – Set aside a portion of the display for the application controls, and gently ensure the user cannot place objects in these regions. This solution provides the user with a sense of stability as the scene is never affected. This approach does cut into some of the available 3D space.
- **Temporarily move the viewpoint** – Gradually move the viewpoint of the scene backwards, or gradually scale the perspective e.g. reducing scene size, affording more foreground space for the UI. When the UI controls are no longer required, gradually return the viewpoint to its previous setting. This approach maximizes 3D space.

In all three approaches, make sure the user understands that these changes are temporary.

Using More Input Devices

Mouse

Although 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.

A zSpace mouse plugin 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. There is no stylus beam, so a Z coordinate helps the mouse pointer move at different depths.

Stylus-based Mouse Emulation

Stylus-based mouse emulation is the ability to imitate the behavior of a mouse using the zSpace stylus. The stylus can be used to drive the operating system mouse cursor. It is almost always better to design your application to use input devices as originally intended—zSpace stylus for manipulating objects in 3D space, mouse for user interface screen input, and keyboard for typing—rather than using mouse emulation. However, if you have legacy applications or must rely on an operating system-provided UI, enabling mouse emulation can be convenient for the user to avoid frequently switching between the stylus and mouse.

If you use mouse emulation in a 3D application, a key challenge is avoiding depth cue conflicts between the pointer and objects in the scene. The mouse pointer should always be on top of objects, which can appear at any depth. That said, using mouse emulation need not be global.

For instance, you can enable and disable mouse emulation dynamically depending on user task:

- If the user is pointing the stylus at the scene or non-legacy UI, disable mouse emulation.
- If the user is pointing the stylus anywhere else, or the legacy UI is visible, enable mouse emulation.
- You can also use a combination of the previous guidelines with a distance-rule, such as the stylus must be within a certain distance of the display to enable mouse emulation.

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.

Trackball

The trackball can be very powerful as a secondary input device. The user holds the stylus in the dominant hand and operates the trackball in the non-dominant hand. The trackball works well in the following 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 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 can help you determine the appropriate scale.

There are some technical issues concerning trackballs. Operating systems generally report a trackball device as a mouse. If the user has both a mouse and a trackball, both devices can control the mouse pointer. Your application can avoid these conflicts as follows:

- When the trackball is in use, hide the mouse pointer.
- When the mouse is in use, display the mouse pointer.
- Offer a keyboard shortcut to switch from the trackball to the mouse.
- When the trackball is active, do not use a standard mouse click to switch devices. In our experience, users accidentally reach for the mouse when intending to use the trackball.

Some trackballs* also distinguish between clockwise and counterclockwise twist of the trackball, presenting them to the system as scroll up / scroll down events. These clockwise / counterclockwise motions can be effectively applied to clockwise / counterclockwise rotation of a selected object as well. Or, this motion can be applied to zoom in / out on the object or scale up / down of the selected object. The best use of this action depends on the application.

* Such as the Kensington SlimBlade Trackball.

Keyboard

This section describes unique combinations of user interface values that involve the keyboard and zSpace interface or the keyboard and stylus.

Keyboard Commands

Creating keyboard commands for a zSpace application allows the user to activate common or necessary commands without moving the stylus out of the workspace. This enables the user to stay engaged with the stylus in their primary activity without losing access to other helpful commands that could be hidden in a menu system.

Switching Stylus Modes: Some unique keyboard commands to consider are those that change how the stylus interacts with an application. In some zSpace applications, the stylus has three distinct modes – *move*, *dissect*, and *stylus cam*. Each gives the user a different ability when interacting with the software using only the stylus. The user never has to disengage from their activity to swap between modes.

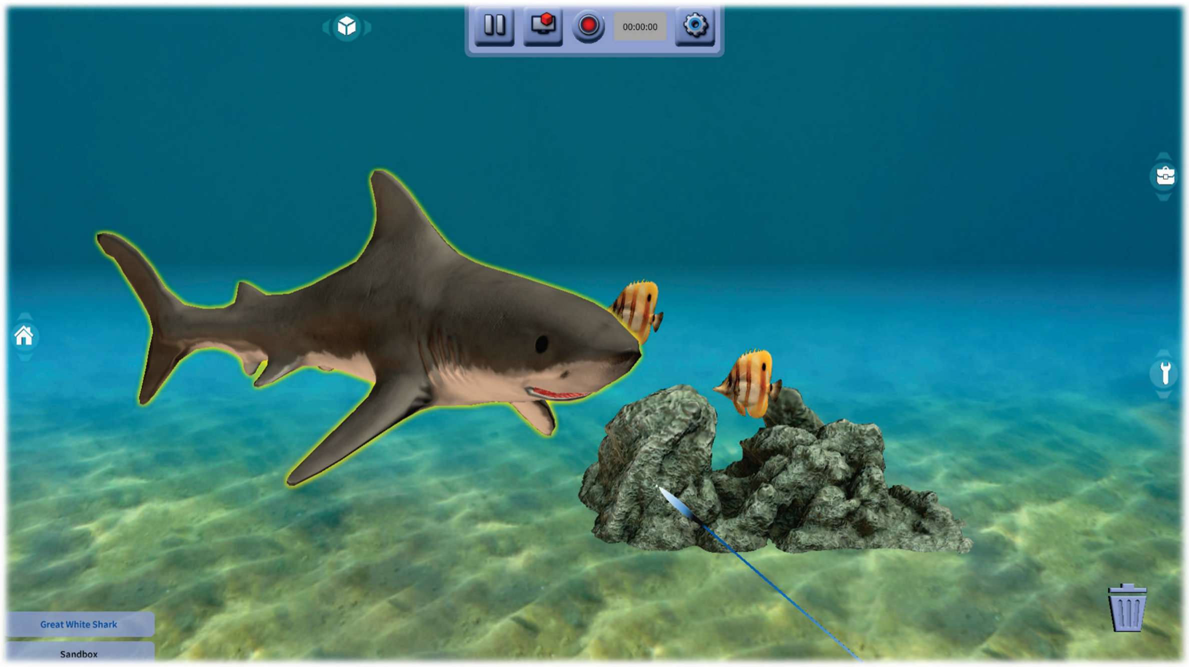
Rotating and Moving Objects or Scene Space: Holding down a button and rotating the stylus can be unintuitive to new users. Using keyboard commands to move or rotate objects, tools, or elements of the scene (including platforms in a scene or the scene space itself) can be more intuitive for new users and convenient for experienced users.

Switching Between 2D and VR/AR Mode: When recording in zView, the user can switch between 2D mode (what a user sees when not wearing zSpace glasses), and VR/AR mode (what the user would see while wearing zSpace glasses). Including a keyboard command to switch between modes helps the user navigate during their recording, without opening a menu or interrupting the recording. The user can also create a more dynamic video in which swapping between viewpoints is a seamless experience.

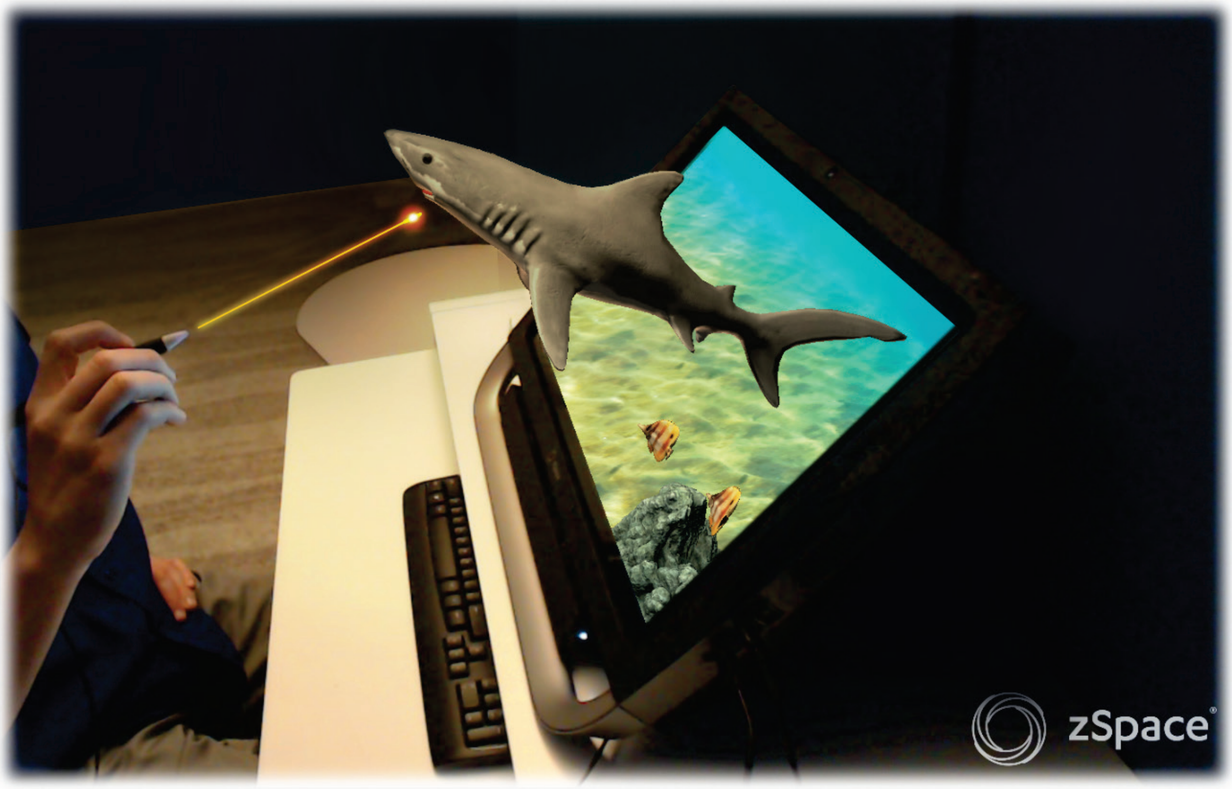
zView Augmented Reality

zView allows users to share their zSpace experience with an audience. Normally, only the person wearing the tracked glasses can see the application in stereoscopic 3D. zView projects the user’s experience onto a screen or a second monitor. You can use zView to share the application in real time, record the zSpace session, or both. zView provides two modes:

- **Standard View** shows the application with head tracking. It adjusts the viewpoint as the user’s head moves. This mode does not require a camera.



- **Augmented Reality View** shows the application from a fixed viewpoint, without head tracking. This mode requires a camera.

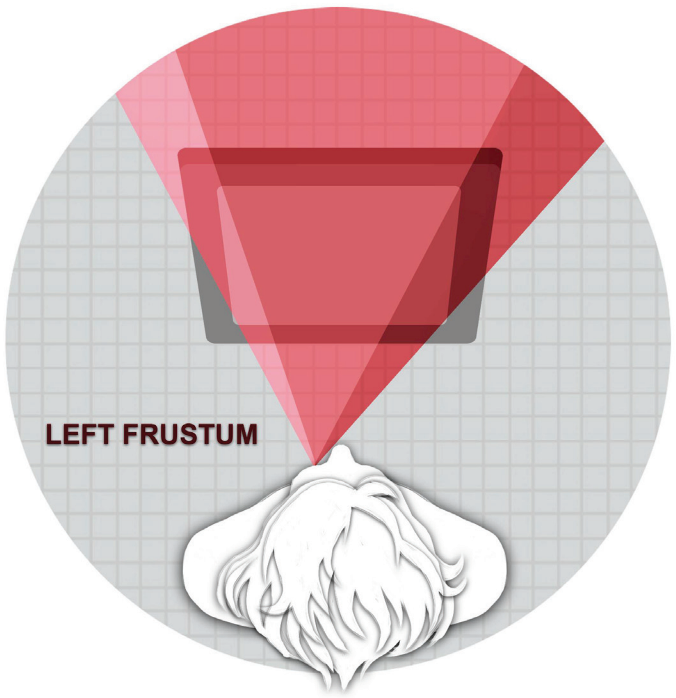


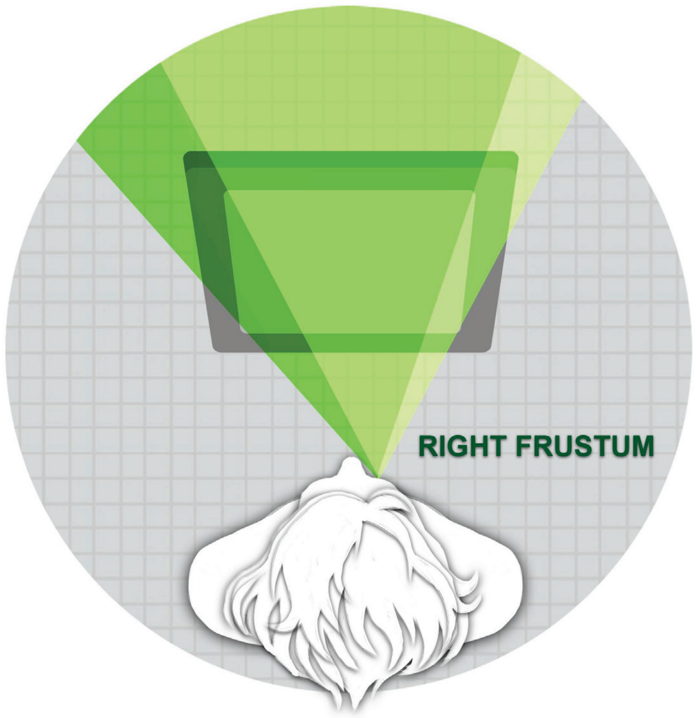
Environment and Workspace

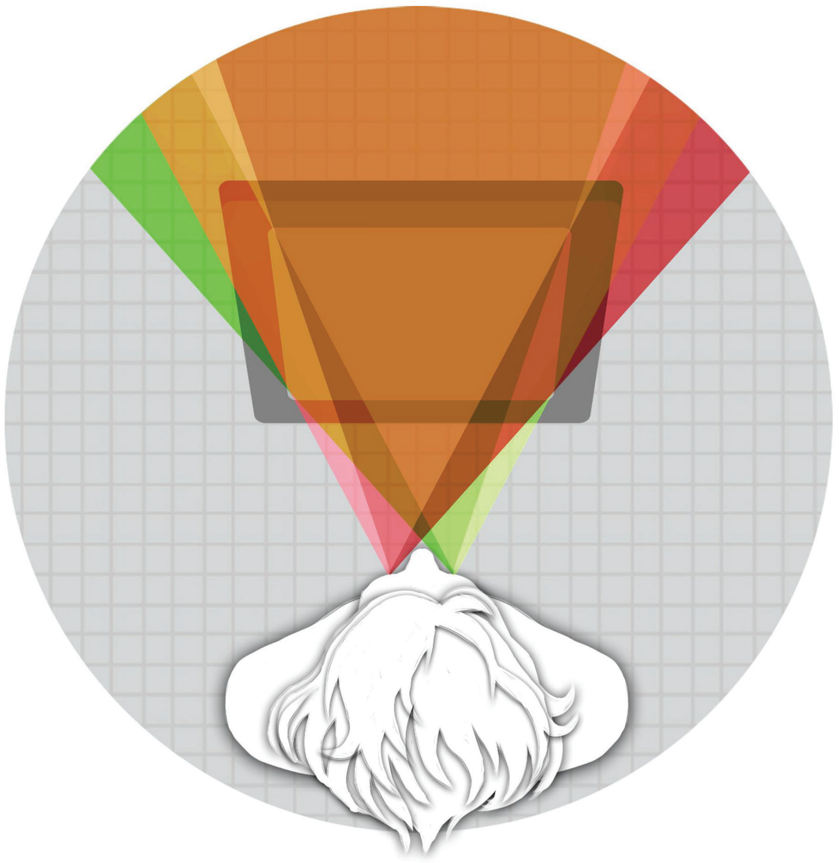
Staging and Composition

zSpace is, at its core, an orthographic stereoscopic experience. *Orthographic* in this context refers to the link between the physical space and virtual space, presenting objects as if they are sharing the same world space as the viewer. zSpace is designed to make this illusion as believable as possible. Anything that breaks the orthographic stereoscopic experience also breaks the illusion of three-dimensional reality.

To satisfy the orthographic condition, the screen must display the perspective of an object as if it is seen by a pair of human eyes. In this case, the field of view is not an artistic choice. It is created by the dimensions of the display connecting to the exact position of the human eyes. Two eyes mean two independent fields of view. As the eyes move in space, the fields of view are recalculated to perfectly match the new eye positions.


LEFT FRUSTUM


RIGHT FRUSTUM

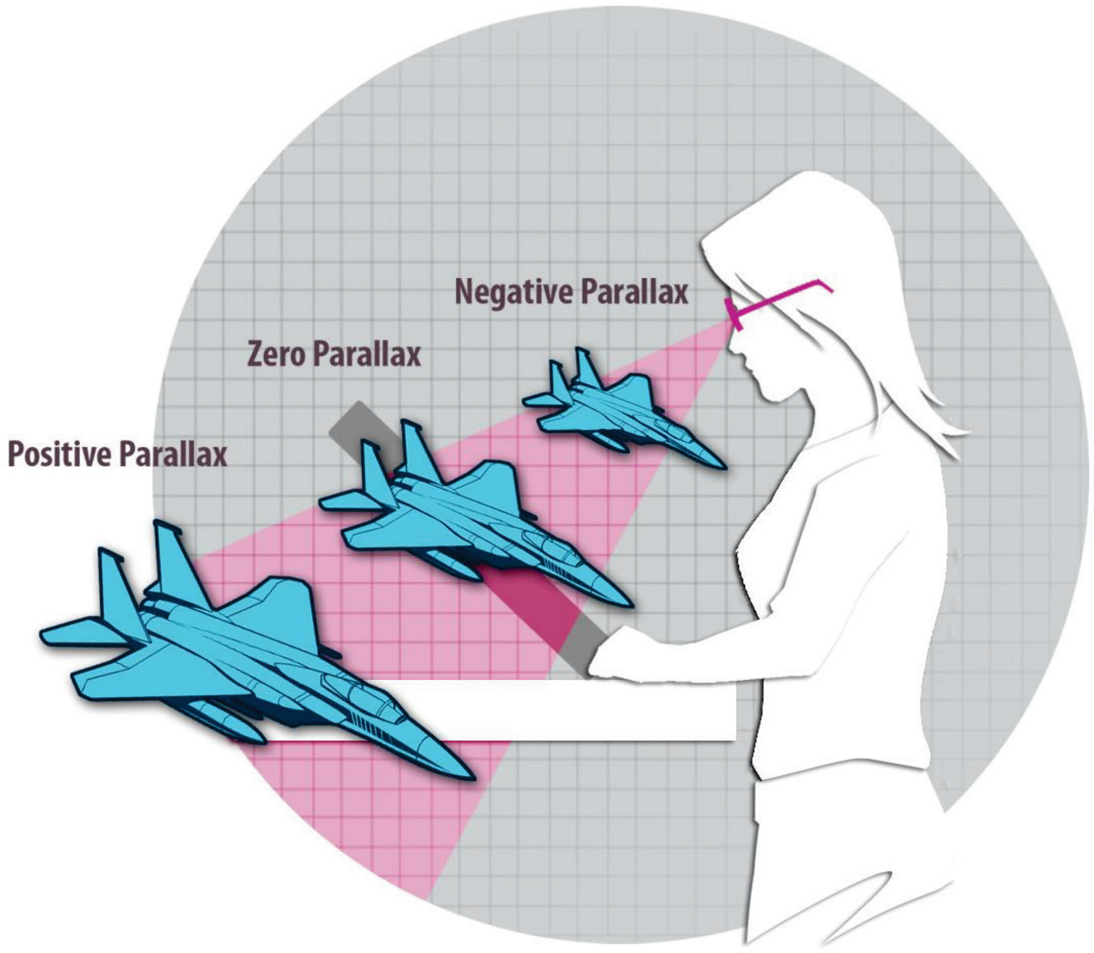


STEREOSCOPIC FRUSTUM

For all of this to work, the tracking of the viewer's eyes, the generation of unique stereoscopic perspectives, the recreation of real world dimensions, and the display timing must be perfectly in sync with each other. This visual combination is an inherent strength of the zSpace platform.

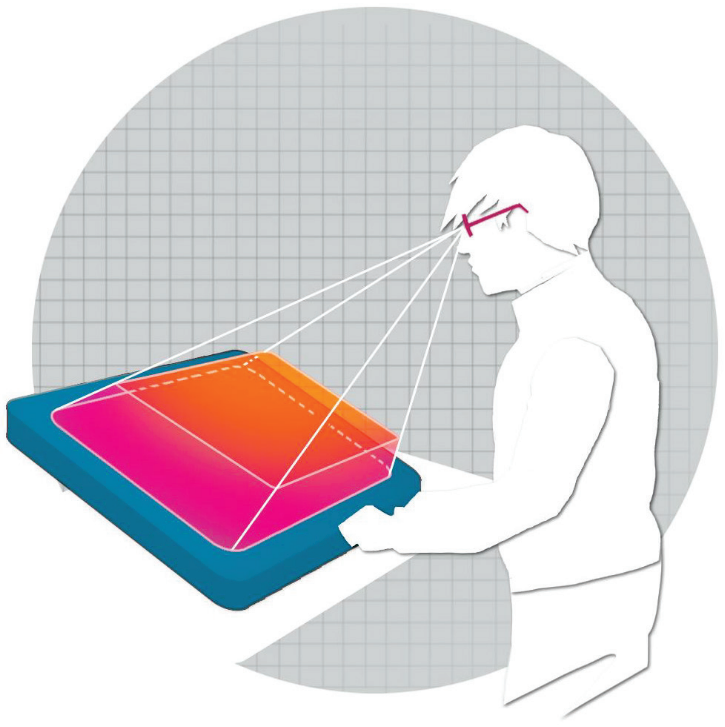
Viewing Zones

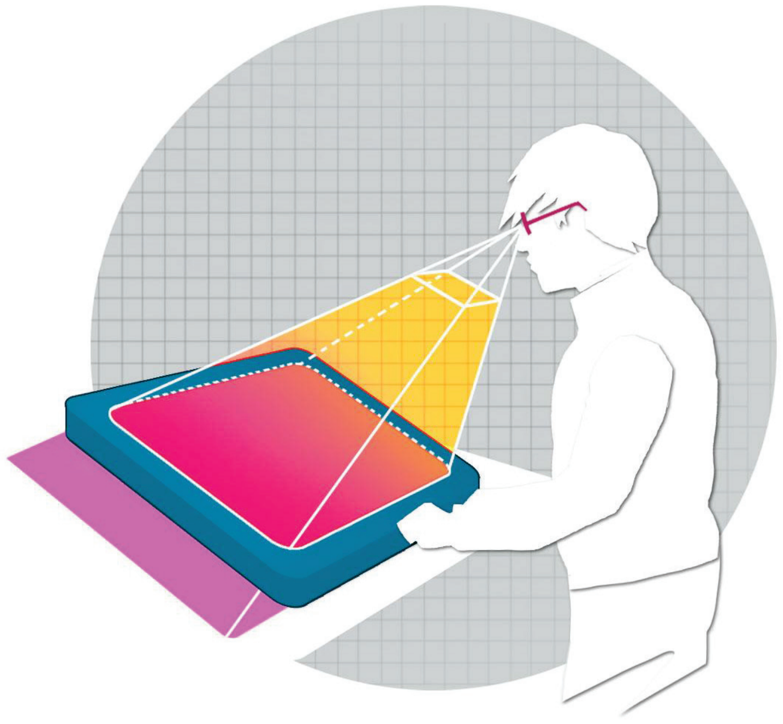
Objects in the zSpace display can appear at different depths. When something appears to be at the exact depth of the screen, or on the screen's surface, it is at *zero parallax*. Objects that appear in front of the screen, projecting towards the viewer, are in *negative parallax*, while objects that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



The zSpace 3D aspect ratio of your application is another consideration. The 1920 x 1080 screen dictates the 2D aspect ratio of the stereoscopic window within the 3D stage. However, the 3D stage has an additional Z dimension that can either be shallow or deep, extend behind the window, in front of the window, or both.

The 3D aspect ratio is defined by the relationship of the viewer's eyes to the screen and forms a pyramid shape projecting out from the viewer's eyes through the corners of the screen and beyond. The pyramid can have a shallow or deep aspect ratio.





The 3D pyramid can move left, right, up, down, in, or out as the viewer's head moves. **No 3D image can exist outside of the pyramid. There is no screen to create the image beyond the 1920 x 1080 dimensions**. This ability to move around creates a virtual space much larger than the physical screen size suggests. Applications should encourage head movement to discover and explore the scene in 3D.

Workspace Bounds

When designing an interactive experience for zSpace, remember that the area in which the user interacts with objects is limited. Object placement and interaction must prevent unintentionally striking the physical display with the stylus or being too deep in positive parallax, rendering the object unreachable. For more information see Rendering Content in Chapter 4. Incorporate solutions for objects that end up outside the view frustum, such as the viewer dropping an object behind their head or punching through the floor/screen. Practice good user interface design to let users undo errant placement or spin the environment around to locate an object.

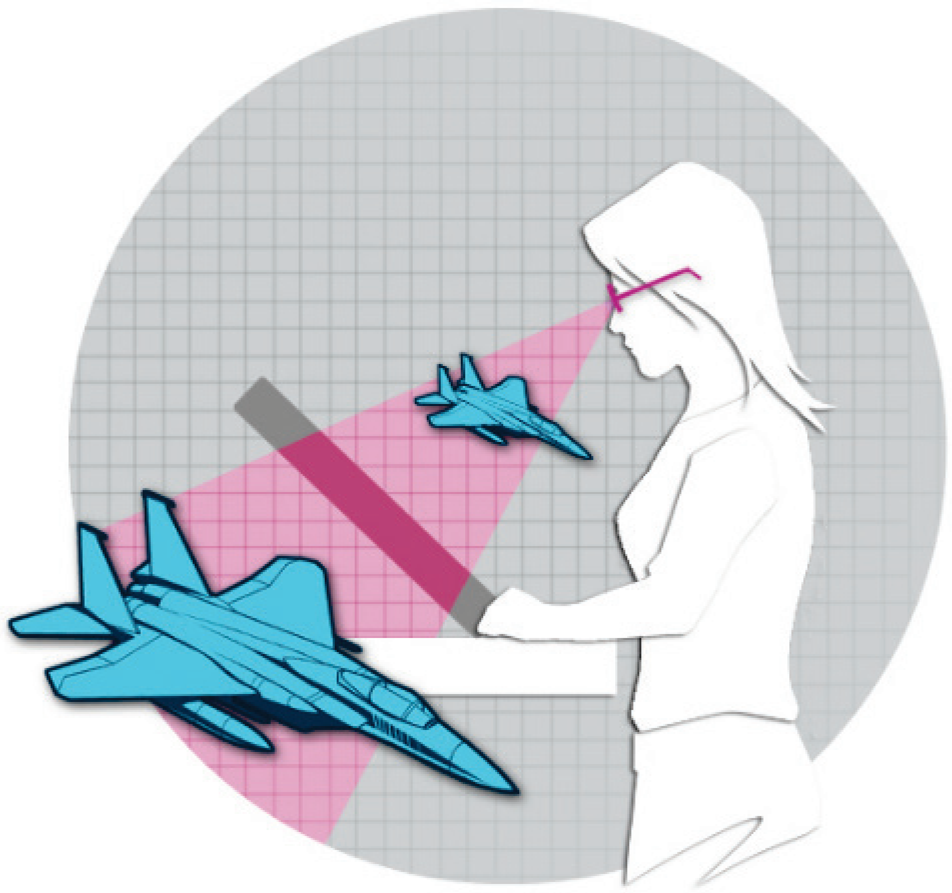
Object Placement and Scale

The large zSpace stage allows for a wide variety of object placement. An object could be to the left or right of the stage, but it can also pop out to the front of the stage or be deep at the back of the stage. Choosing to stay within the coupled zone still allows for a wide range of spatial composition.

Objects that are closer to our eyes naturally draw our attention first. Objects that are distant become secondary or peripheral. zSpace application design can take advantage of this natural behavior by placing objects or instructions of immediate importance closer to the front of the stage. Front of stage placement can also be used to distract or misdirect the viewer while something surprising happens in the background.

As the placement of an object drops to the back of the stage, it can appear less important and more passive to the viewer. It can seem visually out of reach or far away.

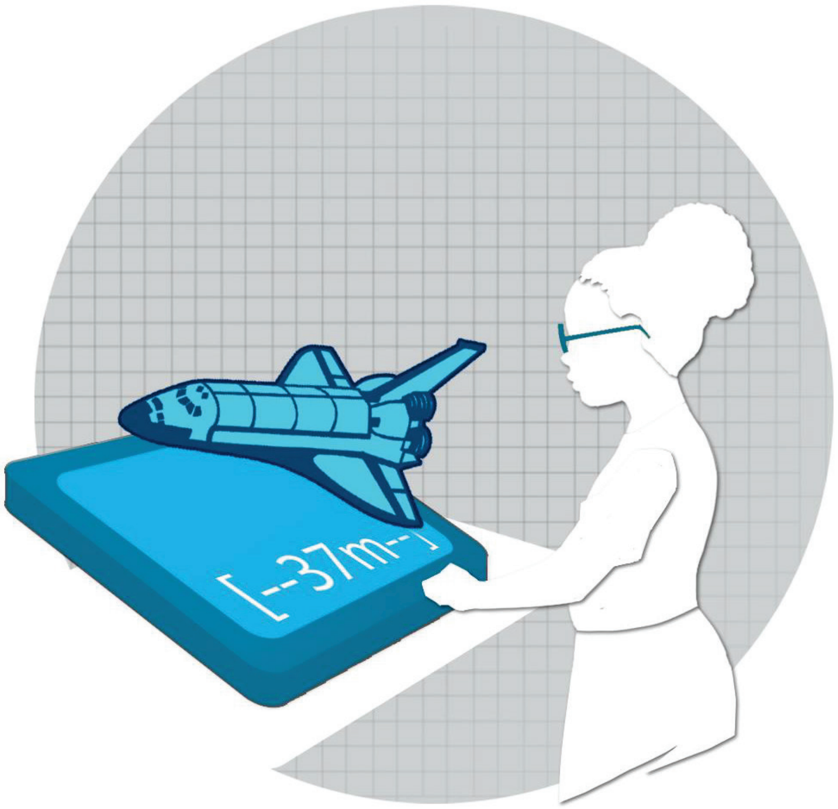
The scale of an object is also affected by the placement within the zSpace stage. Consider two identical objects, of the same size and scale. One of the objects is at the front of the stage. It is contained within the nearer end of the pyramid. It appears small and close, even though it takes up a large part of the user's field of view. The identical object at the back of the zSpace stage is nearing the wide base of the pyramid and appears much larger, even though it takes up less of the user's field of view.



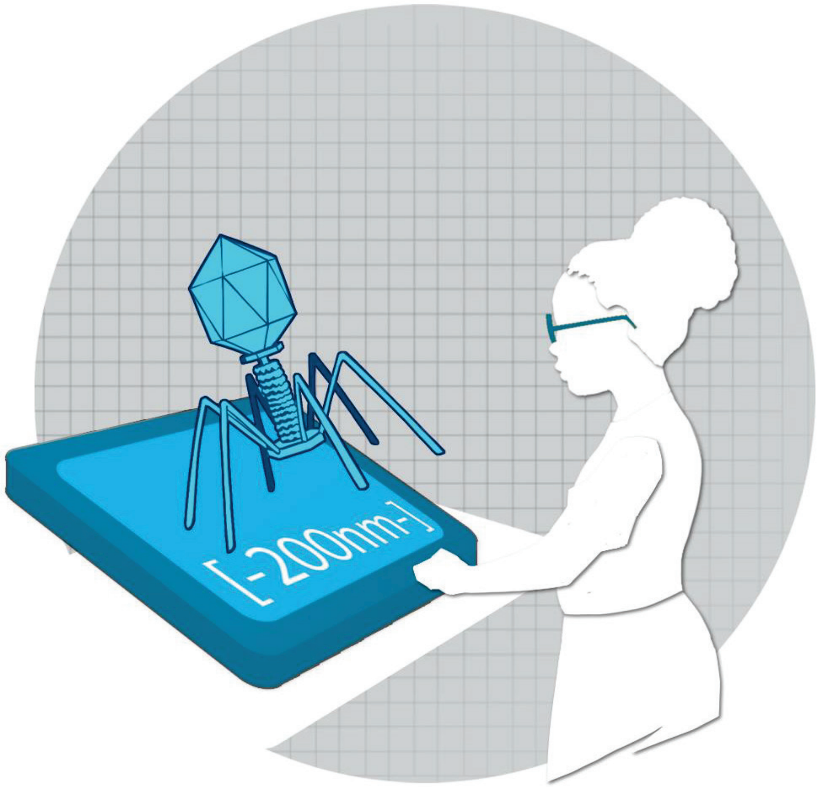
In this example, the X and Y pixel dimensions of the object have not changed. The depth data alone creates the scale illusion. Staging objects closer can make them seem smaller and less intimidating. Staging objects further away can make them appear large and powerful.

Although it is technically possible to recreate a full scale truck or even a solar system within the zSpace stage, it may not be aesthetically desirable. Viewing a distant planet at real world scale on the zSpace system would be like looking through a small window to a very distant and flat space scene at the back of the zSpace stage. It may be technically correct, but it is likely to be uncomfortable and visually disappointing.

Scaling down the solar system and expanding its three-dimensional volume so that it uses more of the zSpace stage is much more interesting to the viewer, giving them a giant's view of the scene. This creates a sense of miniaturization called *hyper-stereo* which can be adjusted to meet aesthetic preferences.



In contrast, a microscopic world would be impossible to see if represented life size. Scaling up the microscopic world to fill the zSpace stage with a hypo-stereo image provides a sense of wonder and exploration.



An application that includes both micro and macro worlds may choose to bias the placement of the micro worlds toward the front of the zSpace stage and macro worlds toward the back. Try to keep both within the comfortable range of the coupled zone or slightly beyond. These placement adjustments create scale differences between the worlds that enhance the viewer's experience and understanding of the information.

Viewer Scale

To assist with rapid design and minimizing art pipeline changes, zSpace includes an optional *viewer scale* adjustment. By default, many zSpace-enabled environments and plugins assume assets and levels are in units of meters. This assumption can be changed by altering viewer scale. A large value means that the user experiences the world as if they are a giant observer, in effect shrinking the true scale of the content. A small value means users experience the world as if they are a tiny observer, in effect enlarging the true scale of the content. For more information about configuring viewer scale, see Chapter 4. Technical Guide.

Angle Awareness

The zSpace system acts as a window into a virtual world, in which certain objects can come through the window. The system tracks this information, incorporates it into the stereoscopic calculations, and makes it available to the application. In the zSpace systems, viewers can adjust the display angle to increase viewing comfort by manipulating the leg stand. We recommend angles between 30 and 60 degrees.

In some zSpace systems, the angle of the display is dynamically tracked and this information can be incorporated into the experience.

- **Angle Awareness Disabled** – If you want 3D application to stay locked to the display, adjust the display angle to rotate the world in tandem.
- **Angle Awareness Enabled** – If you want the virtual world to be locked to the real world, move the display (as a window) independently from the 3D experience.

Designing your experience for disabled angle awareness is simpler, as the framing of your experience does not change. This however can feel odd for applications which use physics and gravity, as the user's display angle may mean virtual gravity no longer points in the same direction as real-world gravity.

Enabling angle awareness means your virtual gravity always aligns to real gravity, however your framing of the experience is now dynamic and may require more design effort. This is a bit like designing web experiences for different form factor devices or display orientations such as portrait vs. landscape.

See the following Chapter 4. Technical Guide for more information about the technical configuration of angle awareness.

Chapter 4:
Technical Guide

Technical Guide

This section describes features and functions of the zSpace platform that are important to developers creating zSpace applications. General concepts include stereoscopic display, head tracking, stylus, and visual performance. Technical concepts include system architecture, native programming, and Unity 3D programming.

General Concepts

zSpace mixed reality platforms feature stereoscopic display, glasses, stylus devices, and tracking systems. This section describes these components and how they work together to deliver the zSpace experience.

Stereoscopic Display

The zSpace display presents images at a rate of 120 Hz and resolution of 1920x1080 per frame. Each sequential frame targets either the left or right eye, in an always alternating fashion. When not wearing the eyewear, both of the viewer's eyes see 120 images per second. When wearing the eyewear, each eye sees 60 images per second, and each eye sees a different set of images. Light is filtered in the eyewear using passive (no electronic components), circularly polarized filters.

At a software level, discussed later, the output buffer is equivalent to a single *stereoscopic* image, and it is the GPU's responsibility to ensure this stereoscopic image buffer is properly converted to 120 Hz with alternating left and right eye image metadata.

On newer display protocols, left and right frame metadata is supported natively. On older display protocols, the zSpace runtime performs a left/right synchronization step when an application is launched.

zSpace displays also work as normal *monoscopic* displays. Most zSpace applications automatically switch from mono to stereoscopic upon detection of glasses or stylus.

Angle Awareness

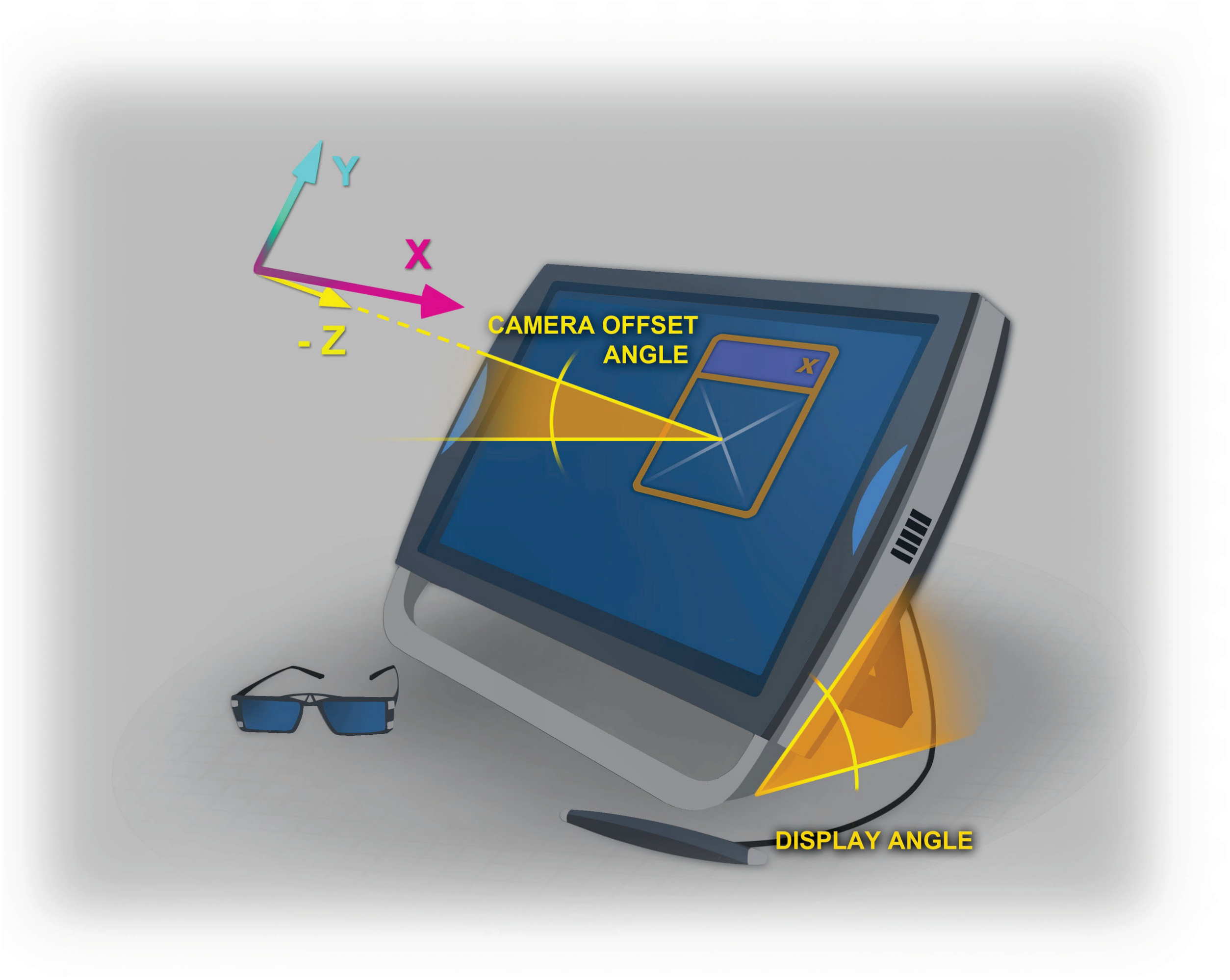
zSpace systems are generally aware of the angle of the display in real-time, which can be used to alter the behavior of the "window." Often this is to align the virtual world's gravity to real world gravity, but angle information is also available for other purposes. Disregarding angle information in effect anchors the virtual world to the physical display, such as rotating the world along with any initial or gradual rotation of the display. See Chapter 3. Best Practices for angle awareness best practices.

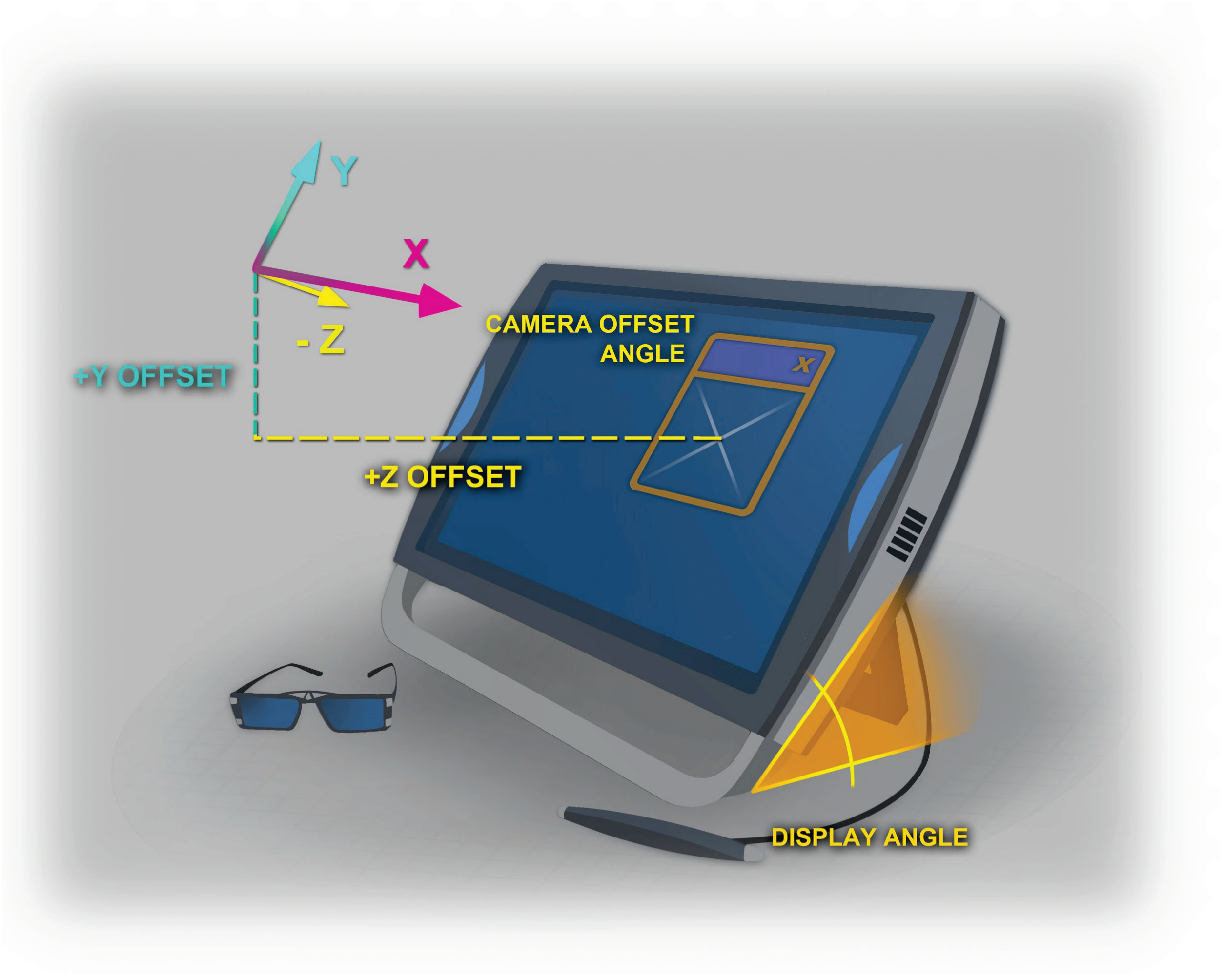
Coordinate Systems

The zSpace system uses coordinate systems for head tracking and stylus tracking. Head tracking is transparent to developers. Stylus tracking requires an understanding of coordinate spaces to enable visualization and use.

Camera Space

Camera space is the coordinate system defined by the virtual camera in the application. Camera space is independent of head tracking. An application needs to position a virtual camera in order to depict a scene and place objects in that scene. The following figure shows how camera space relates to the zSpace display.

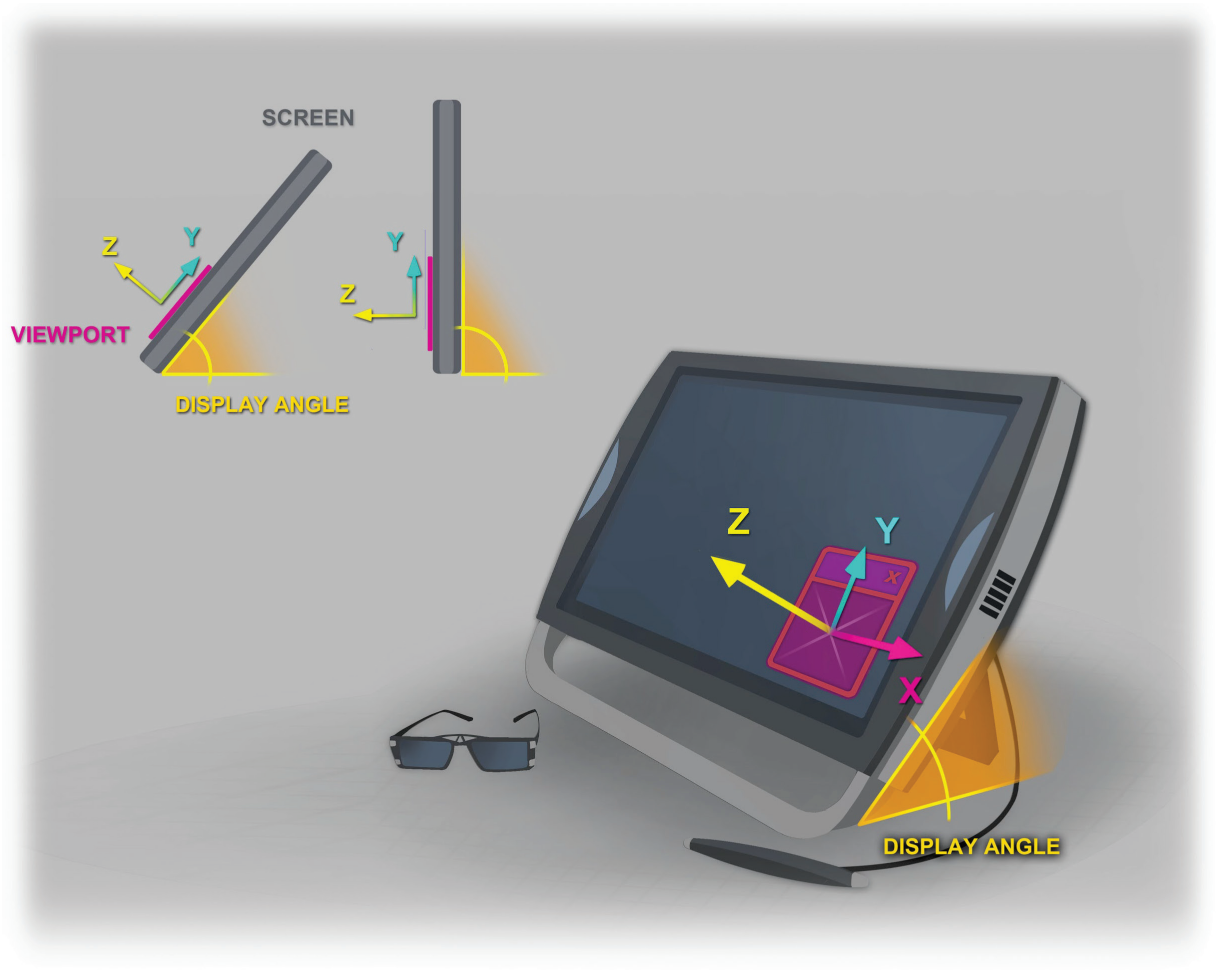




The window that the application renders into is called the *viewport*. The origin of camera space is at the position of the application's virtual camera. The viewport on the zSpace screen is a specific distance away from the virtual camera, and positioned in the direction that the camera is oriented. The virtual camera points to the center of the viewport. If the application is a full-screen application, the virtual camera points to the center of the screen. The distance from the virtual camera to the screen can be calculated by using the zSpace camera offset and display angle. That calculation is presented in the *zSpace Developer Native Programming Guide* and the *zSpace Unity 3D Programming Guide* available at developer.zspace.com. You need to know this distance in order to position an object at or near zero parallax.

Viewport Space

Viewport space is a coordinate system with its origin on the zSpace display. The X axis is parallel to the bottom of the screen and extends to the right. The Y axis is parallel to the edge of the screen and extends up. The Z axis is perpendicular to the screen and extends out of the screen. This is useful for applications. By getting the eye position in viewport space and using the viewport position and size, an application can construct the appropriate starting ray for rendering.



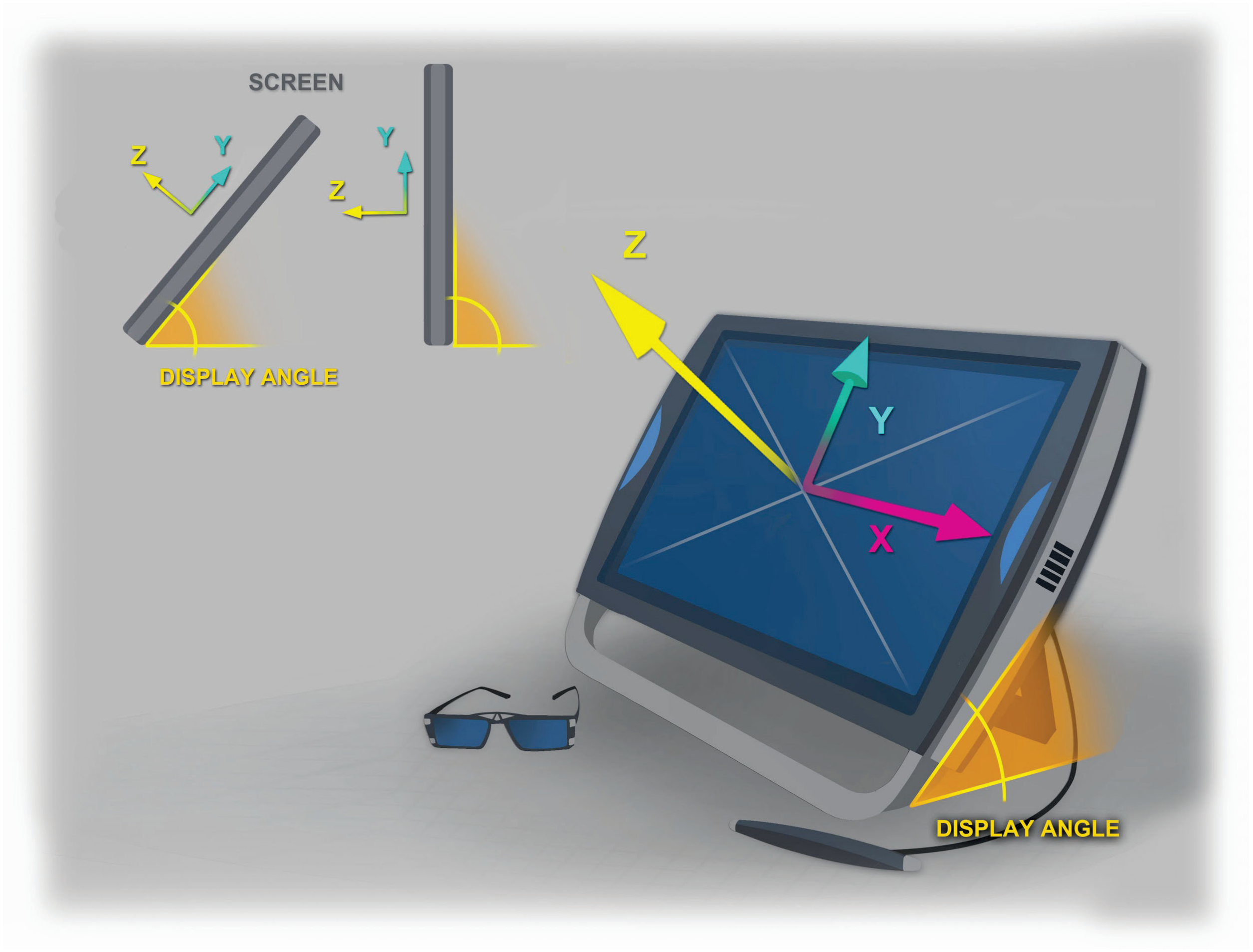
The viewport is the portal into the virtual world. When the viewport is moved on the display, or the display angle changes, there are two ways to react to these events. Either move the world with the viewport or display angle, or keep the virtual world static and move the viewport through the world. Applications control this behavior by adjusting the portal mode used by the frustum. For more information about viewport space and portal mode, see the *zSpace Developer Native Programming Guide* and the *zSpace Unity 3D Programming Guide* available at developer.zspace.com.

World Space

World space is a coordinate system that is only known to the application. zSpace does not need to know how the application has defined the transform from the virtual camera space to application world space. When applications want to use the stylus, they often want to use the stylus in world space. To get any pose in world space, the application must get the pose from zSpace in the camera space, and then transform it into world space using its own internal camera. For an example of how this mapping can be computed automatically, consult the *zSpace Developer Unity 3D Programming Guide* and samples at developer.zspace.com.

Display Space

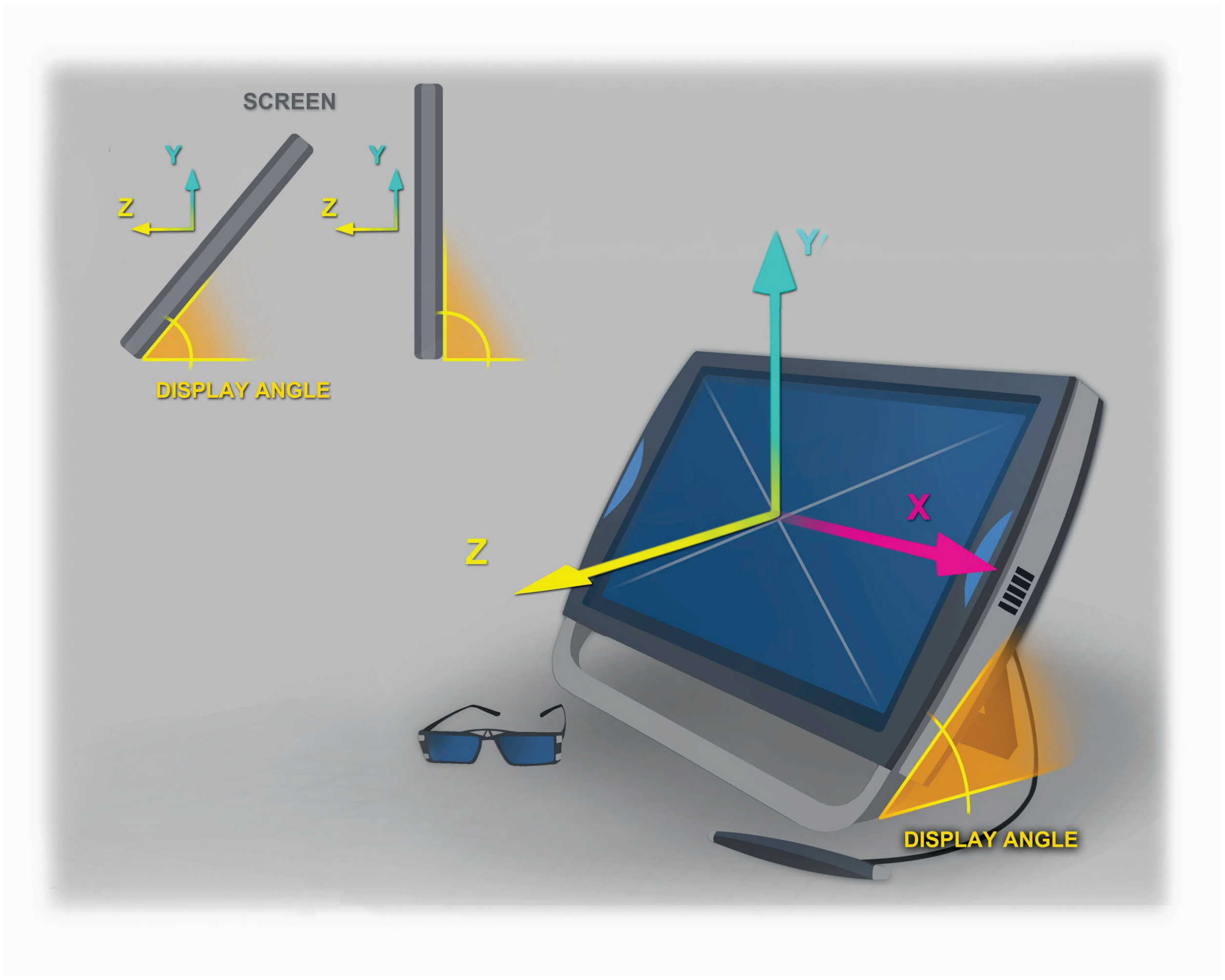
Display space is similar to viewport space, but it is located at the center of the display. If the center of the rendering window is at the center of the display, then they are identical.



The display space orientation is the same as viewport space. When the application is running in full-screen mode, display space and viewport space are also identical. Most applications should use viewport space and do not need to use display space.

Tracker Space

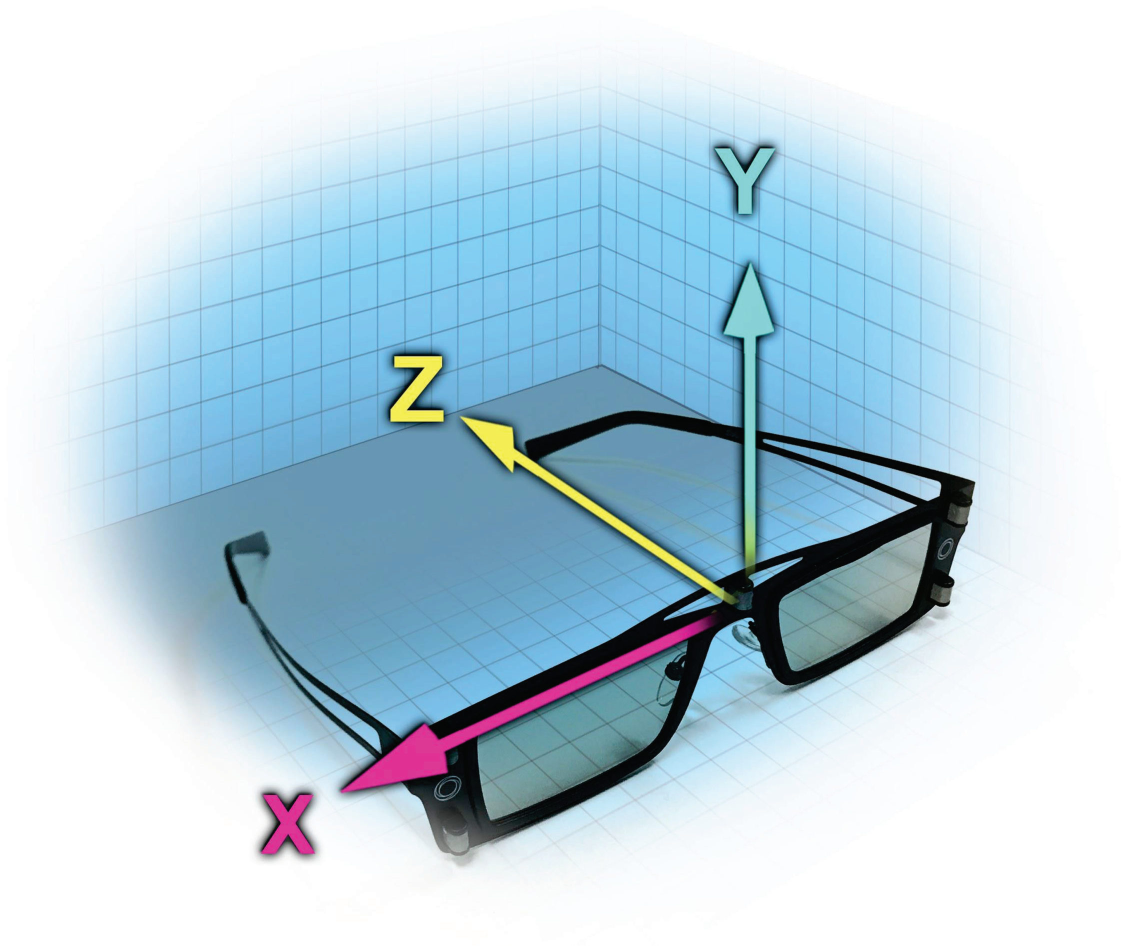
Tracker space is the coordinate system in which raw tracker data is reported. The tracker space origin is the center of the screen. The X axis is parallel to the physical ground and extends along the right of the screen. The Y axis is perpendicular to the physical ground and extends up towards the sky. The Z axis is parallel to the physical ground and extends out of the screen. This coordinate space is used internally, and most applications do not need to use the tracker space.



Head Tracking and Glasses

The zSpace platforms have an integrated system that tracks certain objects. One of the tracked objects is the stereoscopic glasses, which enable users to see into the virtual world. The system calculates the 3D position and orientation of each object, and tracks the objects asynchronously at a rate of at least 100 Hz. This is commonly referred to as an object with six degrees of freedom (6DOF). The data that encodes this information is called a pose.

The pose position is located at the center of the glasses, near the bridge of the nose. The glasses are oriented to a right-handed coordinate system, with the X axis projecting along the right of the glasses, the Y axis projecting up from the glasses, and the Z axis projecting back toward the viewer's head. See the following figure.

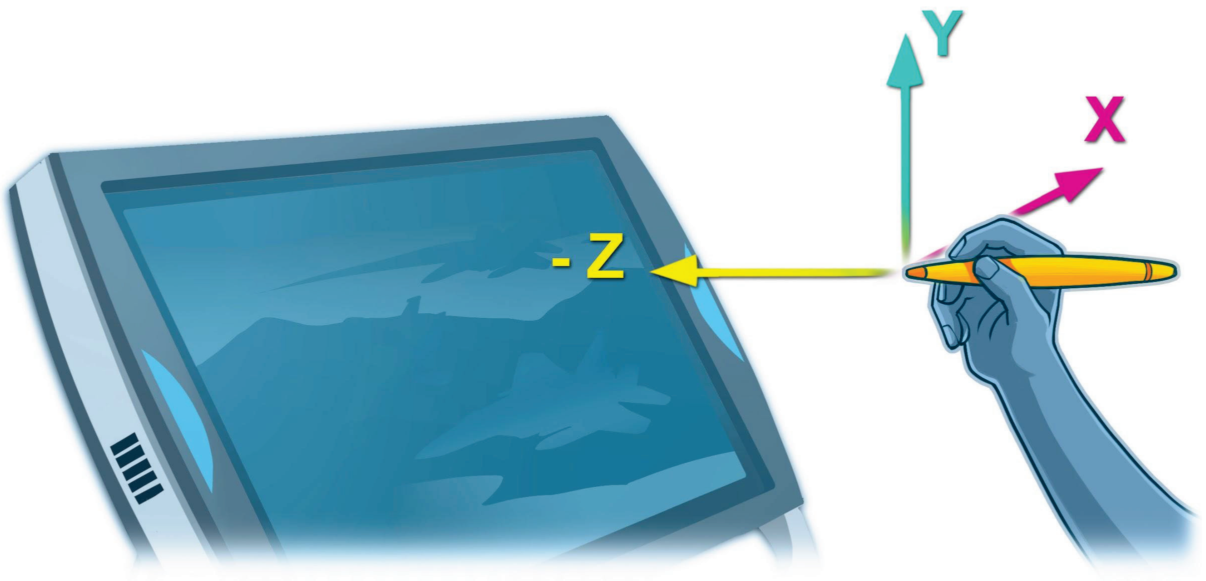


The system can transform this pose position into a number of coordinate spaces. The SDK uses the pose position to provide stereoscopic information to the application. Other minor adjustments are incorporated by default, such as an offset for the typical distance between the user's eyes and the glasses.

For more information on pose data see the *zSpace Developer Native Programming Guide* and the *zSpace Unity 3D Programming Guide* available at developer.zspace.com.

Stylus

The system also tracks the stylus. The pose position is located at the front tip of the stylus. Like the glasses, the stylus is oriented to a right-handed coordinate system. The X axis projects from the right of the tip, the Y axis projects up from the tip, and the Z axis runs back along the stylus. The system can transform the stylus pose into a number of coordinate spaces.



Buttons

The stylus has a number of buttons. An application can query the number of buttons and the state of each button.

LED

At the center of the stylus is an LED light. The SDK can control this light. It can be turned on and off, and set to any red, green, blue combination. For example, if you set all three colors to on, the LED displays a white light.

Vibration

The stylus has a built in vibration capability that can provide some simple haptic feedback to the user. The vibration can be turned on and off, and the vibration waveform is programmed using *on*-duration, *off*-duration, and *repeat count.*

Tap

The stylus can also detect a tap event if the stylus is tapped on the screen.

Enable Mouse Emulation

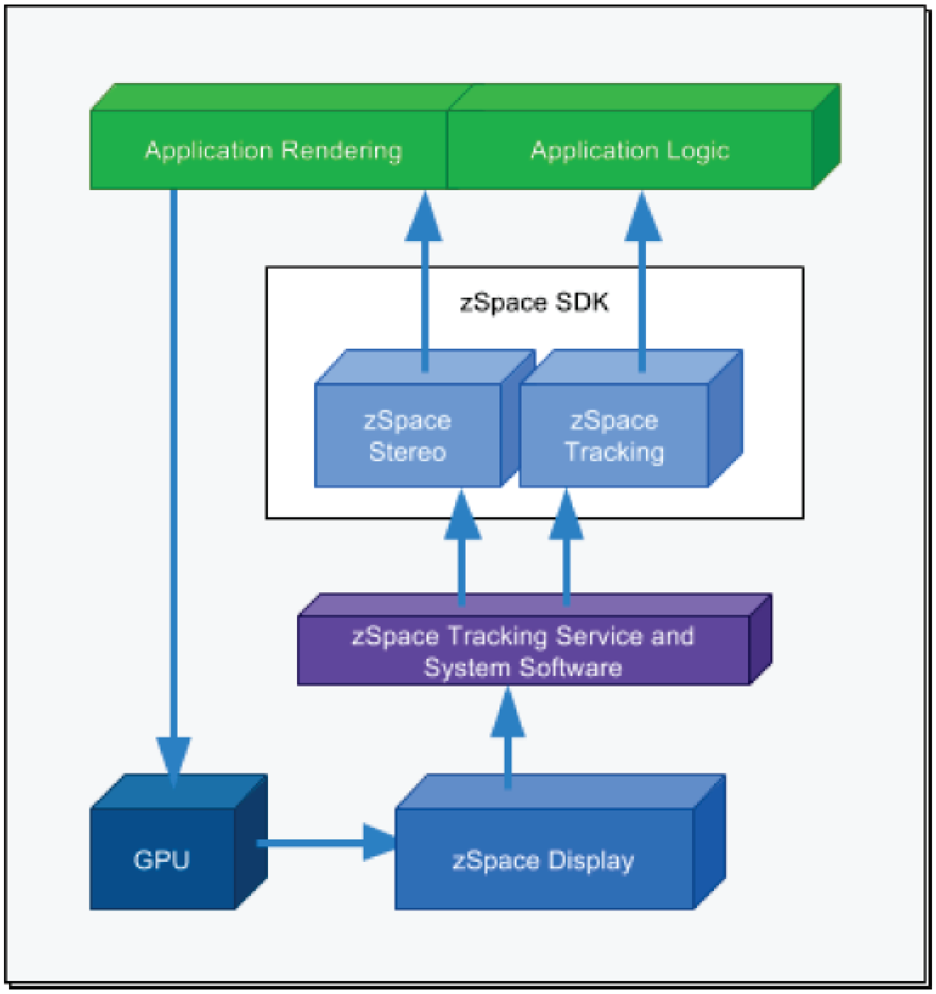
Many applications have existing user interface elements that can be interacted with using the mouse. If you imagine the virtual ray emanating from the end of the stylus and follow it to where it intersects with the display, this is a very natural point for the 2D mouse to exist. Combine that with the three buttons on the stylus, and it is easy to emulate a mouse with the stylus. For more information about best practices of mouse emulation see Chapter 3. Best Practices.

Enable Mouse Auto-Hide

Generally the mouse cursor is visible in applications unless you specifically hide it. This is sometimes needed when interacting with 2D user interface elements. It can also be distracting in a stereoscopic environment. Mouse auto-hide, if enabled, automatically shows or hides the mouse cursor based on recency of mouse activity.

System Architecture

The zSpace SDK system architecture is shown in the following figure.



For tracking, most of the computation happens in the zSpace display. Those results are passed to the zSpace System Software through a single USB connection to the computer. Some computation occurs on the computer—through the SDK—but the system load for this processing is negligible.

While this high level overview represents all zSpace systems and applications with the same APIs, there are differences between the systems, which are articulated in the next section.

Rendering Content

Buffering

Graphics APIs supporting stereoscopic output generally support a *quad buffered* mode, which is the stereoscopic version of double buffering. In this mode, both the “front” and “back” buffers are stereoscopic images. It is the application’s responsibility to render left and right eye information to the back buffer, and to indicate when to flip buffers.

Depth and Clipping Planes

For a detailed introduction to parallax and viewer comfort, see Chapter 2. Mixed Reality Essentials and Chapter 3. Best Practices.

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 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.

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.

In monoscopic applications, the near and far clip planes are generally straight-forward to edit. In stereoscopic rendering, choosing appropriate clipping planes can be more sensitive. You can adjust stereoscopic clipping planes with the help of the zSpace SDK Core libraries.

Follow these guidelines for setting your clipping planes:

- Set the near clipping plane to greater than half the interpupillary distance, which is the distance between the viewer's pupils. By default, the interpupillary distance is 0.06m.
- As with monoscopic rendering, make the ratio between the far and near clipping planes as small as sufficient for your needs. This reduces the likelihood of z-fighting. Note that in stereoscopic rendering, z-fighting results can differ between each eye at any given instant, affecting stereoscopic comfort.

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.

World and Object Scale

Scale is related to a coordinate system. All 3D applications use a scale, even if implicitly. Fundamentally, zSpace does all processing in the real world physical scale. To achieve the desired experience, you need to be aware of the relationships between the zSpace scale and the application 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 *viewer scale*. As an example, if your world is 100 times larger than the zSpace display, use a viewer scale of 100. In that case, objects that appear to be 75m 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 depends on the size of your objects.

On the other hand, if your application needs to display microscopic detail, you would set a smaller viewer scale so objects appear larger than in the real world. Choosing the correct world scale depends on what you are modeling. To present objects at their real-world size, simply use 1.0 viewer scale.

Adjustments to scale are possible programmatically via the *viewer scale* setting. For an example in greater detail, consider the display may be 0.521 meters wide, suitable to display a half-meter wide object at 1:1 scale. However, if the viewer scale is set to 10.0, then the effective screen width would be 5.21 meters, suitable to display a 5 meter wide object at 1/10 scale. Applications that render very large or very small objects using true units should use *viewer scale* to quickly "port" their coordinate system scale to physical units for zSpace.

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

Vertical Synchronization (VSync)

Vertical Synchronization (VSync) is a general capability of GPUs in which the application may decide when presenting a new back buffer to the display between doing so during the display's blanking period (VSync *enabled*), or doing so unrestricted or unsynchronized, potentially during the display's physical update (*disabled*).

If VSync is disabled, this results in scene "tearing" for both monoscopic and stereoscopic applications. The tearing effect can be disconcerting in mono but sometimes appropriate for the application. However, in stereoscopic views it causes more severe user discomfort, due to each eye seeing inconsistent tearing between each image. Enable VSync for zSpace applications. This may slow down FPS performance, but it results in a better overall user experience.

General Performance Tips

Performance refers to the system performance such as CPU speed and frames per second (FPS). Responsiveness affects the user perception of performance. This measures response time, or how quickly the application responds to user input. A low FPS rate can reduce or eliminate head tracking benefits to stereoscopic comfort. When the scene refreshes too slowly for the user's head movements, this is worse than no head tracking.

Basic guidelines for achieving suitable performance are as follows:

- Optimize and simplify application logic. If using a rendering library or game engine, know the scalability characteristics and limitations of the technology or middleware.
- Make sure your foreground or rendering thread consistently has low latency to allow the renderer to maintain a high frame rate. Avoid heavy processing in these threads.
- Count the number of draw calls. You may be missing opportunities to render in batches.
- Count the total number of polygons. You may have too many objects in the scene.
- Check lighting. Disable options to see if any single change has a large impact: number of lights, type of lights, lighting effects.
- Avoid complex shaders and materials, such as those with transparency, refraction, and reflection unless needed.
- Check the texture size. Larger or higher resolution textures can slow performance.
- Use simple physics colliders (box or convex shape) where unnoticeable to the user.

REMINDER: Stereoscopic 3D applications must render the scene twice, requiring much more graphics processing power than monoscopic applications.

Improving Responsiveness

In zSpace applications, responding quickly to stylus movements is critical. It is more important to show an object's movements as the stylus moves than to show the complete object, thus where a tradeoff can be made, try displaying lower level-of-detail objects during movement, for faster rendering during the user's actions.

Native Programming

Building or porting an application to the zSpace platform at its lowest level of integration involves two distinct parts. First, the application must be stereo enabled and use the zSpace stereoscopic values to generate correct head tracked images. Second, the application needs to get stylus information into a coordinate system. After those two essentials are established, other aspects of the zSpace platform can be developed to create unique experiences, including viewer scaling, stylus vibration, and mouse emulation.

Software Development Kit

When you download and install the zSpace SDK from developer.zspace.com/downloads, it includes the following components:

- Header files used by the zSpace system.
- Libraries needed for the supported architecture.
- The default zSpace icon.
- Sample programs that show how to use various features of the SDK. These samples are used throughout this document, as code references and complete samples, which demonstrate how to use zSpace features.

The zSpace SDK currently supports programming in C, or C-compatible languages. Additional language bindings for the zSpace SDK are planned for the future, along with full documentation.

Display Buffer Setup

This section describes at a high-level how to allocate the buffers for *OpenGL* and *Direct3D* APIs, and how to modify the normal rendering loop to render into stereo buffers. For further information, refer to each API's respective documentation.

OpenGL

OpenGL applications allocate driver resources by using the *SetPixelFormat* Windows function. Existing OpenGL applications already call this function to allocate resources for the back buffer, stencil buffer, and depth buffer, and simply need to augment the parameters here to enable quad buffering. Refer to the OpenGL documentation or zSpace SDK Samples for specific steps.

Direct3D

Direct3D recently provided a vendor neutral API for quad buffered stereo support in version 11.1. While it is possible to set up quad buffered stereo in Direct3D 9, 10, and 11, it requires functions specific to AMD and nVidia each. Please refer to the **BasicStereoD3DSample** for details about setting up quad buffered stereo on AMD and nVidia for Direct3D 9, 10, and 11.0.

To initialize stereo in Direct3D 11.1, you must create the device and swap chain. There are several steps in this process, and some changes are needed to ensure quad buffered stereo support. The **BasicStereoD3D11_1Sample** depicts quad buffered stereo support.

Stereoscopic Rendering

Most 3D applications start by setting up all of their rendering resources, and then move into a loop, which continuously renders the scene until the application exits. For each iteration of the loop, the code updates the application state, makes the OpenGL context current, clears the back buffer, draws the scene, and swaps the back buffer to the front. This continues until the application exits.

For stereoscopic rendering, the scene needs to be rendered twice, once for the left eye and once for the right eye. The scene is drawn twice, and the draw buffer is set to the appropriate value depending on which eye is being rendered. The view and projection matrix values are different for each eye, so they need to be recalculated per frame for each eye.

Unity 3D Programming

The zSpace Unity Plugin is an easy way to add head-tracked stereoscopic 3D capabilities to your new or existing Unity-based application.

The zSpace plugin supports both Unity 4 and Unity 5. The plugin also supports both the 32 and 64 bit Unity editor and players. To enable stereoscopic rendering in the editor and in builds, generally you must also include a command-line argument when launching (see links to documentation in the following content).

Note: The zSpace Unity Plugin is supported on select versions of Unity 5.x. For the most up-to-date information about Unity 5 support, see zSpace Unity Plugin Releases.

Note: A binary patch is required to support Unity versions before 4.5. Download a patch for Unity 4.3.0 or 4.1.3 from zSpace Unity Plugin Releases.

For implementation details see zSpace Support Unity Plugin Releases.

For more information about Unity development see the Unity 3D Manual.

zSpace Package and Getting Up and Running

The zSpace plugin is distributed as a Unity package and contains a number of files, including plugin DLLs, C# script files, and easy to use *prefabs*.

A **zCore prefab** is a drag & drop part of the zSpace plugin. Import your Unity package then drag in an instance of the zCore prefab. For basic scenes this is enough to support head tracking. You are up and running.

If your scene requires complex scripting and level construction, continue to the following sections for details about the zSpace Unity Plugin.

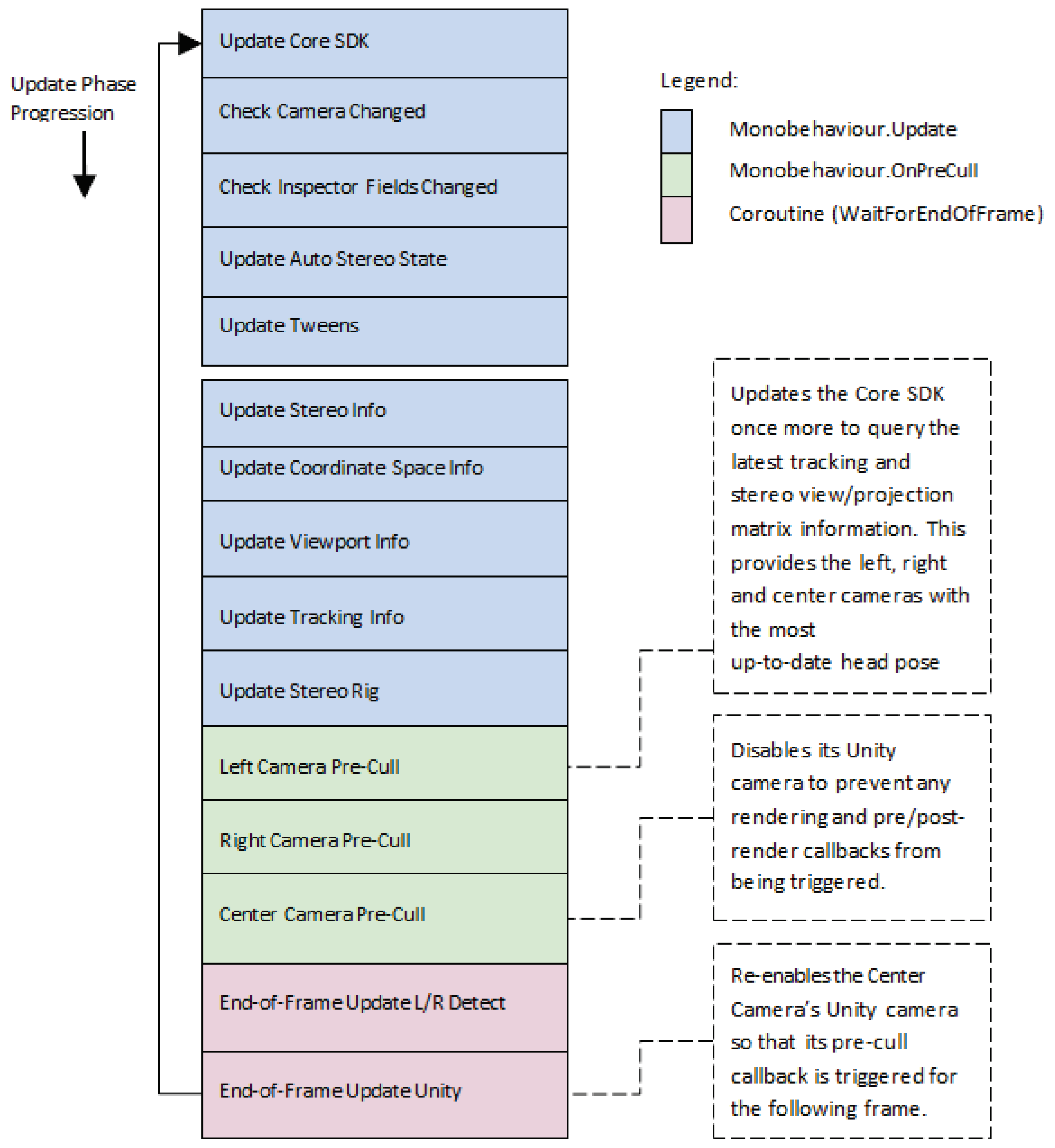
zSpace Plugin Architecture

The Unity developer does not need to modify anything in the zSpace plugin to access all of its functions. But it is helpful to understand the architecture and implementation when building applications. This section explores the plugin architecture.

StereoRig Game Object

The root of the stereo rig hierarchy is the StereoRig game object. StereoRig includes three objects: LeftCamera, RightCamera, and CenterCamera. Each object contains a Unity Camera component and renders the scene for its respective eye. The CenterCamera does no rendering; instead it simply defines the post stereo rendering state, and serves as camera location for any raycast operations.

The following operations occur for every frame in the plugin.


Update Phase Progression
Update Core SDK
Check Camera Changed
Check Inspector Fields Changed
Update Auto Stereo State
Update Tweens
Update Stereo Info
Update Coordinate Space Info
Update Viewport Info
Update Tracking Info
Update Stereo Rig
Left Camera Pre-Cull
Right Camera Pre-Cull
Center Camera Pre-Cull
End-of-Frame Update L/R Detect
End-of-Frame Update Unity
Legend:
Monobehaviour.Update
Monobehaviour.OnPreCull
Coroutine (WaitForEndOfFrame)
Updates the Core SDK once more to query the latest tracking and stereo view/projection matrix information. This provides the left, right and center cameras with the most up-to-date head pose
Disables its Unity camera to prevent any rendering and pre/post-render callbacks from being triggered.
Re-enables the Center Camera's Unity camera so that its pre-cull callback is triggered for the following frame.

Plugin Operations – This Happens for Every Frame in the Plugin

For more information about Unity's order of events, see the Unity Manual Execution Order of Event Functions.

Plugin Callback Events

The zSpace plugin relies on Unity camera callbacks to put the rendering buffers into the proper state for stereo rendering. Applications may also want to be notified when these events occur. To be notified of these events after zSpace has done its processing, there are three events available for listening: PreCull, PreRender, and PostRender.

Maintaining Camera Scripts and Effects

The zSpace plugin disables the Current Camera in the zCore object when rendering, so any custom scripts on that camera are not executed. Applications should add any of these custom scripts (like Effects) to both the LeftCamera and RightCamera for the scripts to execute correctly in a zSpace stereoscopic environment.

Editor UI Properties and Debug Information

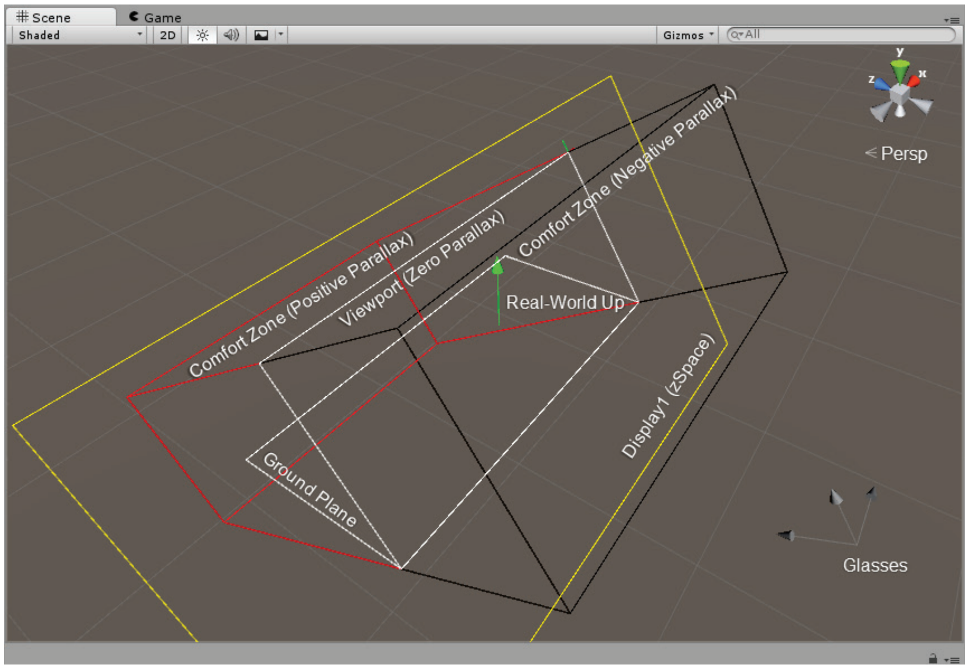
The zSpace plugin has a number of features that are accessible from the Unity editor including settable properties, debug visualizations, and real time data from the tracking system.

zCore Inspector

The main tool for getting data about zSpace execution is the inspector pane of the zCore object. This inspector pane has a number of sections, and each section is described here.

Debug

The Debug section contains a number of checkboxes which control the debug visualizations that are visible in the Scene window of the Unity editor. These visualizations update dynamically, and are present in both edit mode and play mode. The following figure shows an example of these visualizations.



Debug Visualizations

Available options are:

- Show Labels
- Show Viewport (Zero Parallax)
- Show Comfort Zone (Negative Parallax)
- Show Comfort Zone (Positive Parallax)
- Show Display
- Show Real-World Up
- Show Ground Plane
- Show Glasses
- Show Stylus

Stereo Rig

The Stereo Rig section of the inspector controls properties of zSpace stereoscopic viewing. The properties and functions are:

- Current Camera
- Update Current Camera Transform
- Enable Stereo
- Enable Auto-Transition to Mono
- Copy Current Camera Attributes
- Minimize Latency
- Interpupillary Distance (IPD)
- Viewer Scale
- Auto Stereo Delay and Duration

Glasses

The glasses section of the inspector provides real time data about the glasses. Data includes:

- Tracker-Space Pose
- World-Space Pose

Stylus

The stylus section of the inspector provides real time data about the stylus, and checkboxes to enable some stylus features. See the Stylus Data and Controls in zCore Inspector section for more information.

Samples

Stylus Game Object Manipulation

A very common operation in zSpace applications is to pick up and directly manipulate objects with the stylus. The **StylusObjectManipulationSample** scene and script show how to do this.

The sample describes how to compute the world rotation for the object. Then compute the current offset from the end of the stylus to the end of the object. This is the **_initialGrabOffset** variable. Maintain this offset as the stylus moves and cache the cumulative rotation of the stylus and the object in the **_initialGrabRotation** variable. This is the starting rotation to use when adding new stylus rotations.

The **StylusObjectManipulationSample** scene and script are pre-built and included with the zSpace Core SDK.

System Setup and Diagnostics

This section describes zSpace system setup for development.

System and GPU Requirements

When developing on the zSpace 100, 200, or HP Zvr systems, your system must meet certain requirements. See the requirements list at support.zspace.com/hc/en-us/articles/204780665-zSpace-System-Requirements.

zSpace displays require specific GPU and GPU Driver support. For more information see support.zspace.com/hc/en-us/articles/204780645-zSpace-Supported-Graphics-Cards-GPUs-.

System Check and Control Panel

Once you have set up your computer and connected the zSpace display, you can run the system check program. For the zSpace system software 4.3 and earlier, follow the steps at support.zspace.com/hc/en-us/articles/204780485-zSpace-System-Check-and-Control-Panel-Walkthrough.

For the zSpace system software 4.4.3 and later, follow the steps at support.zspace.com/hc/en-us/articles/205737709-zSpace-Control-Panel-and-System-Check.

zSpace 200 systems can use the 4.4.3 system software.

Demo Applications

Once your system is up and running, you can find a variety of demo applications at

support.zspace.com/hc/en-us/sections/201194579-Demo-Applications.

Chapter 5: Partners

zSpace Partners

This chapter describes brand guidelines, STEM user interface, and STEM content creation.

Brand Guidelines

Follow these specifications to identify zSpace products and services.

Use zSpace branding marks only with the express permission of zSpace, Inc. Use only the logo that accurately represents your relationship with zSpace.

Guidelines for displaying the zSpace name and logos:

- Do not display zSpace branding as a primary or prominent feature on any non-zSpace materials. When displaying a zSpace mark, companies must also display their own logo(s), business name, product names, or other branding in the primary and more prominent position.
- Do not imitate or use the zSpace brand marks as a design feature in any manner.
- Do not use the zSpace brand marks in a manner that disparages zSpace or its products or services.
- Do not alter zSpace branding in any way, including changes in the color, proportion, design, or removal of any words, artwork, or trademark symbols. Do not animate, morph, or otherwise distort the zSpace brand marks.

In the zSpace visual system, there are a few core brand elements. Each element is designed to work in harmony with the others and, when combined, convey the zSpace brand identity.

Name Usage

When used in normal text, *zSpace* is always written as such. There is no space between **z** and **Space**. The **z** is always lowercase. The **S** is always uppercase. Use of the zSpace wordmark is described in a following section.

Color Palette

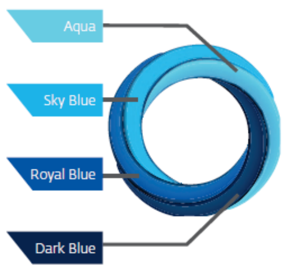
Color plays an important role in communicating the personality of the zSpace brand. Using the brand color palette consistently strengthens brand awareness and recognition.

The primary colors in the zSpace brand color palette include four blues, black, white, and two grays. These colors play a prominent role in all company communications and are the colors used in the brand marks.

The secondary colors are one purple, violet, red, and orange. These secondary colors broaden the color palette and support the primary colors. You can use these colors as an accent; do not use them as the primary color in any design or as the backdrop for brand marks.

Use the Pantone colors whenever possible. Use the CMYK, RGB, and Hex values as a last resort when the usage or application does not accept Pantone colors.

Color Specifications



Primary Brand Colors		zSpace Brand Color	Pantone®	CMYK	RGB	HEX
	◢	zSpace Dark Blue	Pantone 648 C	100/68/0/54	0/45/98	#002D62
	◢	zSpace Royal Blue	Pantone 300 C	96/69/0/0	0/92/170	#005CAB
	◢	zSpace Sky Blue	Pantone 306 C	69/7/0/0	17/181/233	#13B5EA
	◢	zSpace Aqua	Pantone 310 C	51/0/9/0	112/204/226	#70CDE3
	◢	Black	Solid Coated Black C	0/0/0/100	0/0/0	#353132
	◺	White		0/0/0/0	255/255/255	#FFFFFF
	◢	zSpace Dark Gray	Solid Matte 425 M	0/0/0/77	95/96/98	#5F6062
Secondary Brand Colors	◢	zSpace Light Gray	Solid Matte Cool Gray 3 M	0/0/0/17	216/217/218	#D8D9DA
	🟪	zSpace Purple	Solid Coated 2597 C	85/100/0/0	82/46/143	#522E91
	🟥	zSpace Violet	Solid Coated 221 C	24/97/50/6	182/41/89	#B62959
	🟥	zSpace Red	Solid Coated 188 C	24/100/100/22	158/28/32	#9E1C20
	🟧	zSpace Orange	Solid Coated 159 C	10/68/95/0	223/112/46	#DF702E

Brand Marks

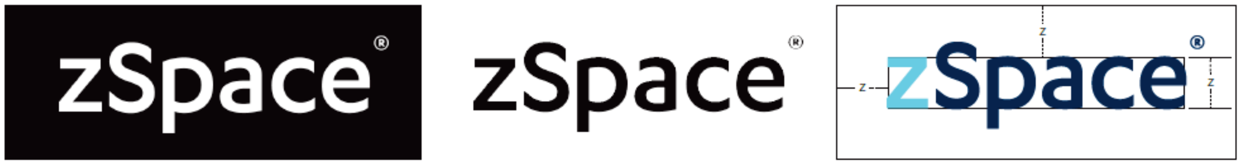
The zSpace brand marks include a wordmark, portal symbol, horizontal logo, and vertical logo.

Use the color versions on a white background only. For all other backgrounds, use a monochromatic version.

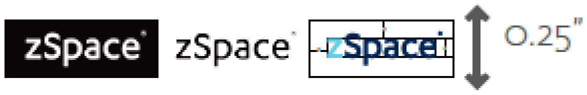
Keep the wordmark, portal, and logo clearly visible and visually separate from other content. No graphics, type, photography, or illustration should violate the mark's clear space on any side. This clear space is derived from the height of the lower-case letter **z** in the zSpace mark. The following images show the required clear space.

Wordmark

Use the zSpace wordmark to denote the zSpace product, but not the zSpace platform. The register symbol ® is part of the wordmark and may not be removed.

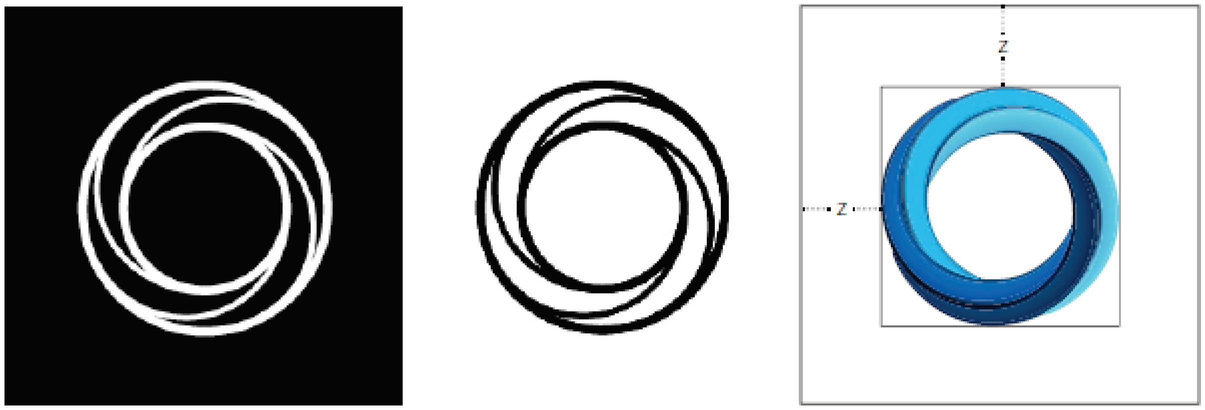


To ensure legibility, the zSpace wordmark must be 0.25” in height or larger.

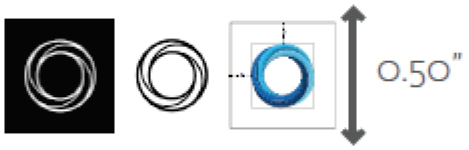


Portal

The portal is a symbol of the zSpace platform’s ability to connect users with a new experience. Use the portal only to depict the zSpace platform. Do not use the portal alone to represent the zSpace product.



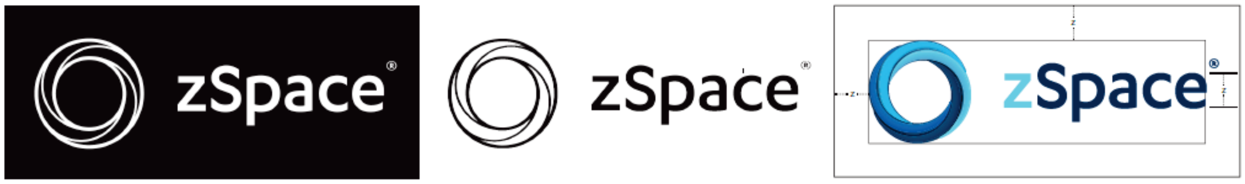
The portal must be 0.50” in height or larger.



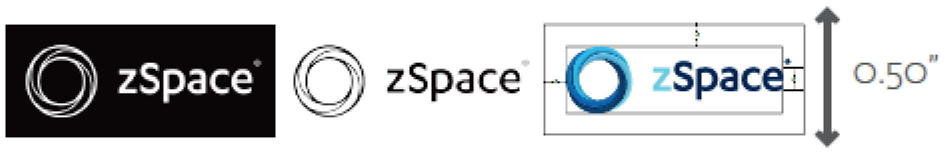
Logo

The logo is a combination of the portal and the wordmark. Use it to denote the zSpace platform or experience, but not the zSpace product. The register symbol ® is part of the logo and may not be removed.

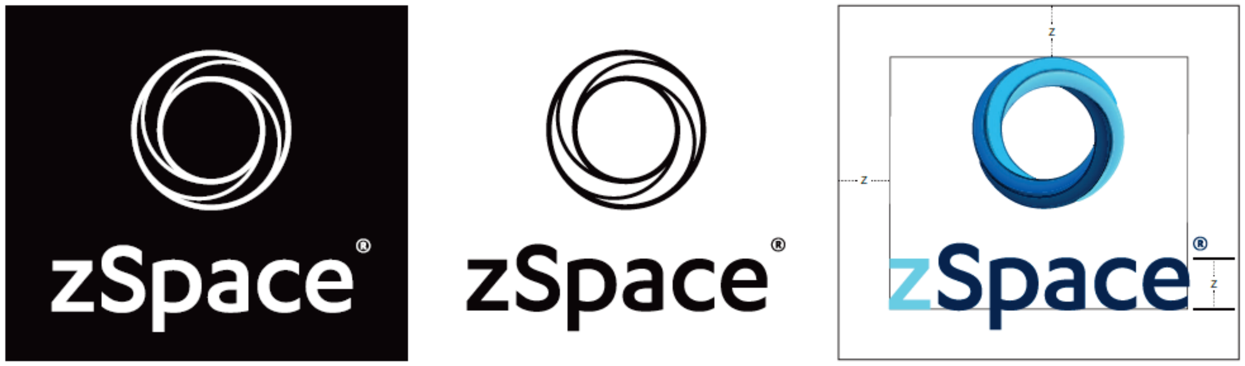
Horizontal Logo



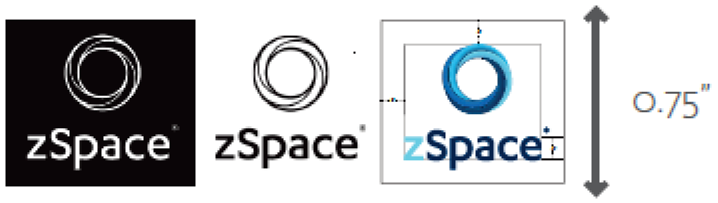
The horizontal logo must always be 0.50” in height or larger.



Vertical Logo

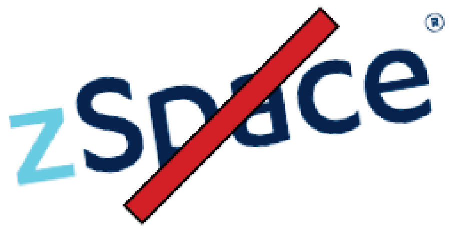


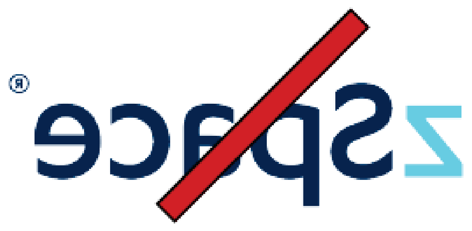
The vertical logo must always be 0.75” in height or larger.



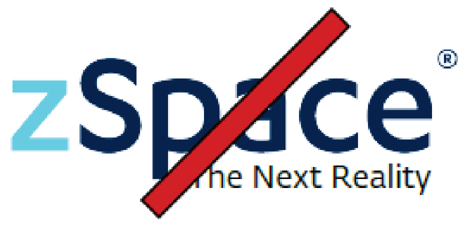
Brand Mark Misuse

Do not rotate or flip the logo.

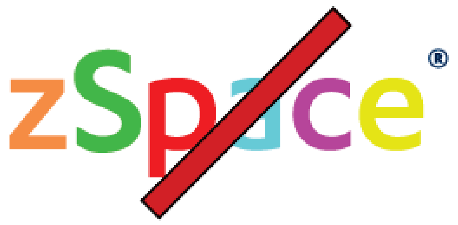


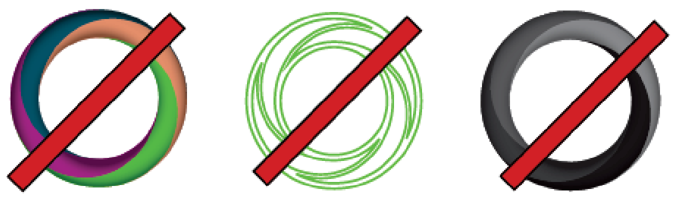


Do not violate the logo clear space.

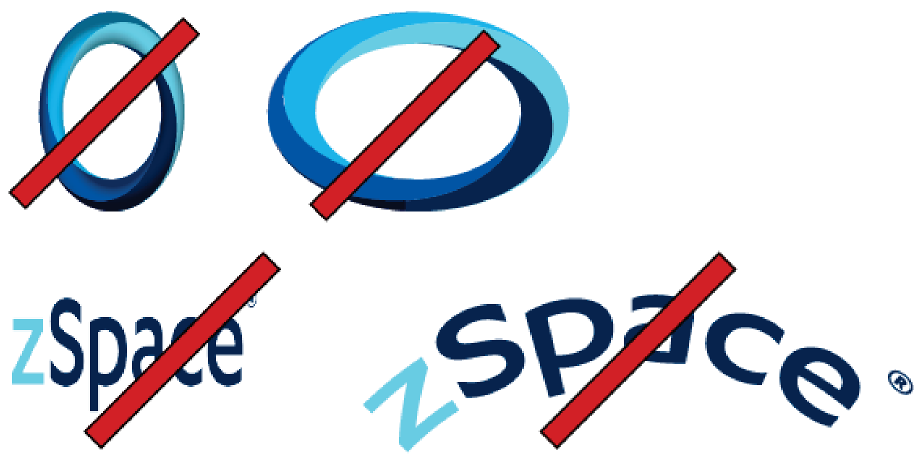


Do not change any logo colors.



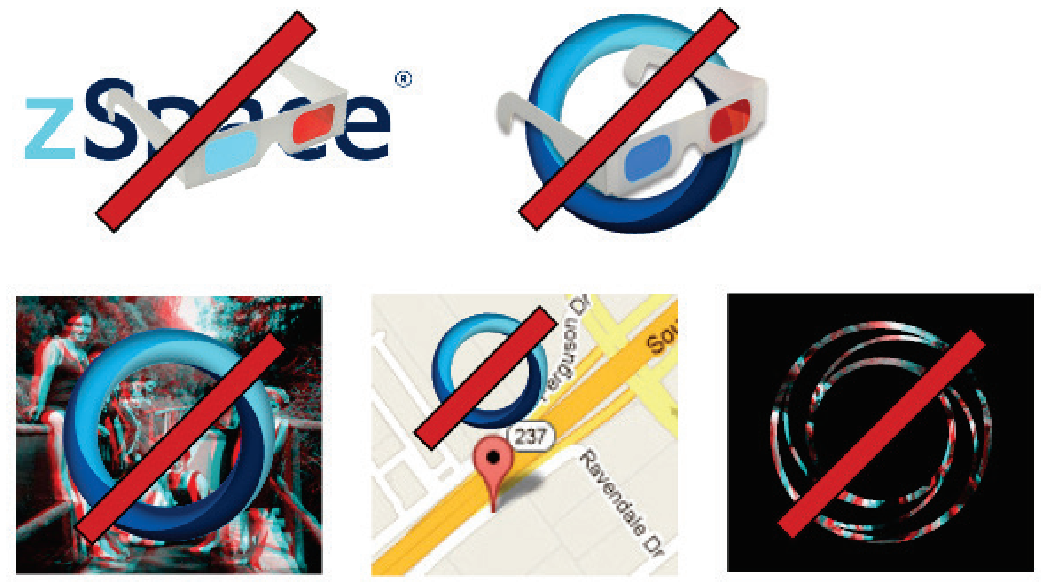


Do not skew, stretch, or otherwise change the logo proportions.

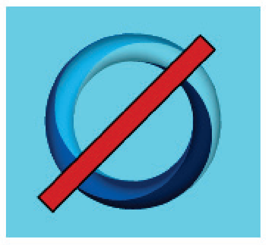


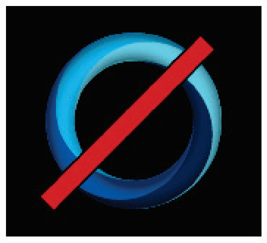
Do not add text to the logo.

Do not place the logo on images if the background interferes with the logo legibility.



For color backgrounds, use only monochromatic logos.





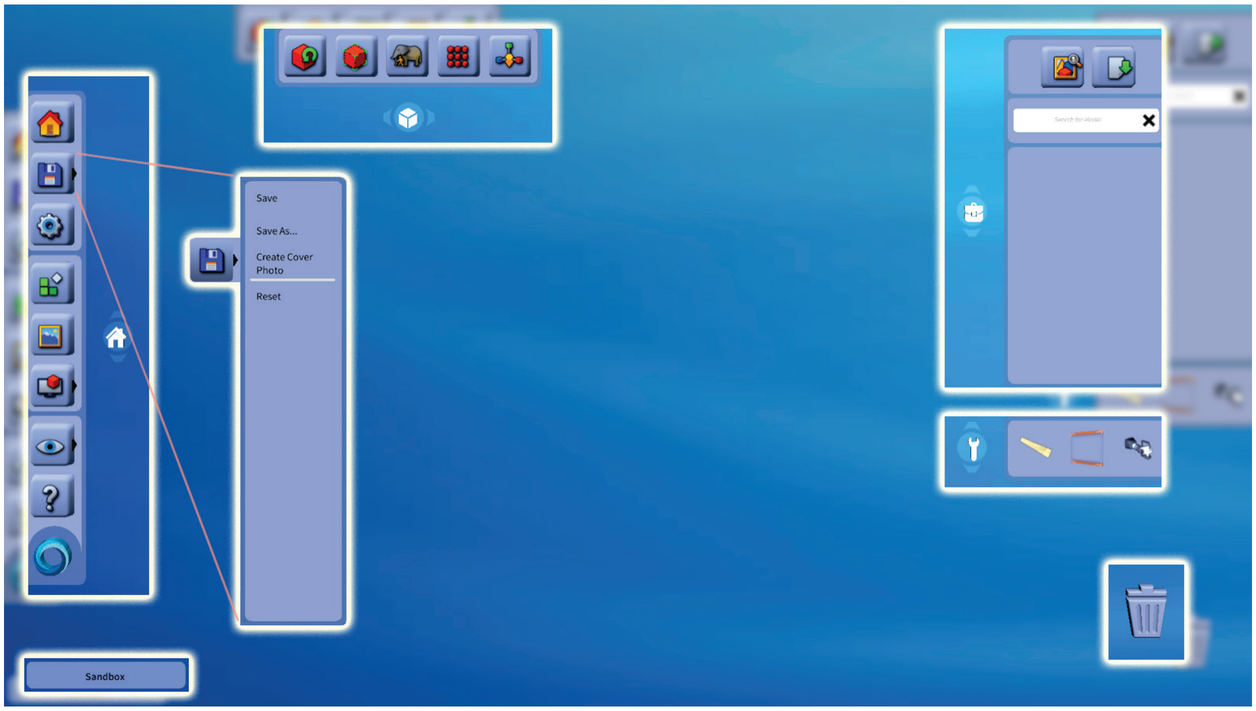
Do not use the Portal as a bullet point.

zSpace is a revolutionary, immersive, interactive 3D environment for computing, creating, communication, and entertainment.

zSpace offers powerful new capabilities and provides a competitive edge in business, education, and government applications.

STEM User Interface

The zSpace educational software suite features a common user interface for most applications. This includes a consistent look and feel and a set of standard menus, toolbars, and dialogs.



Overview of STEM UI Placement

Note: In zSpace Studio, the Control Bar is on the left, with submenu option. The activity name is displayed in the bottom left corner. Additional application-specific commands appear at the top of the screen. The backpack (shown empty, with search function) and the tool palette are on the right. The trash can appears at the bottom.

Control Bar

The application control bar contains all of the application-level commands. It should contain main commands such as file operations, preferences, and exiting the application.

The application control bar appears on the left side of the screen. The control bar can be dragged to the right side of the screen and docked there as well. This gives the user some flexibility on where they prefer to see their tools and menus; for instance, left-handed users may have different preferences than right-handed users.

Some icons on the control bar are linked to a submenu that appears when the icon is clicked. These submenus provide access to basic operations that could be grouped under that icon. For example, the save submenu includes **Save** and **Save As** options.

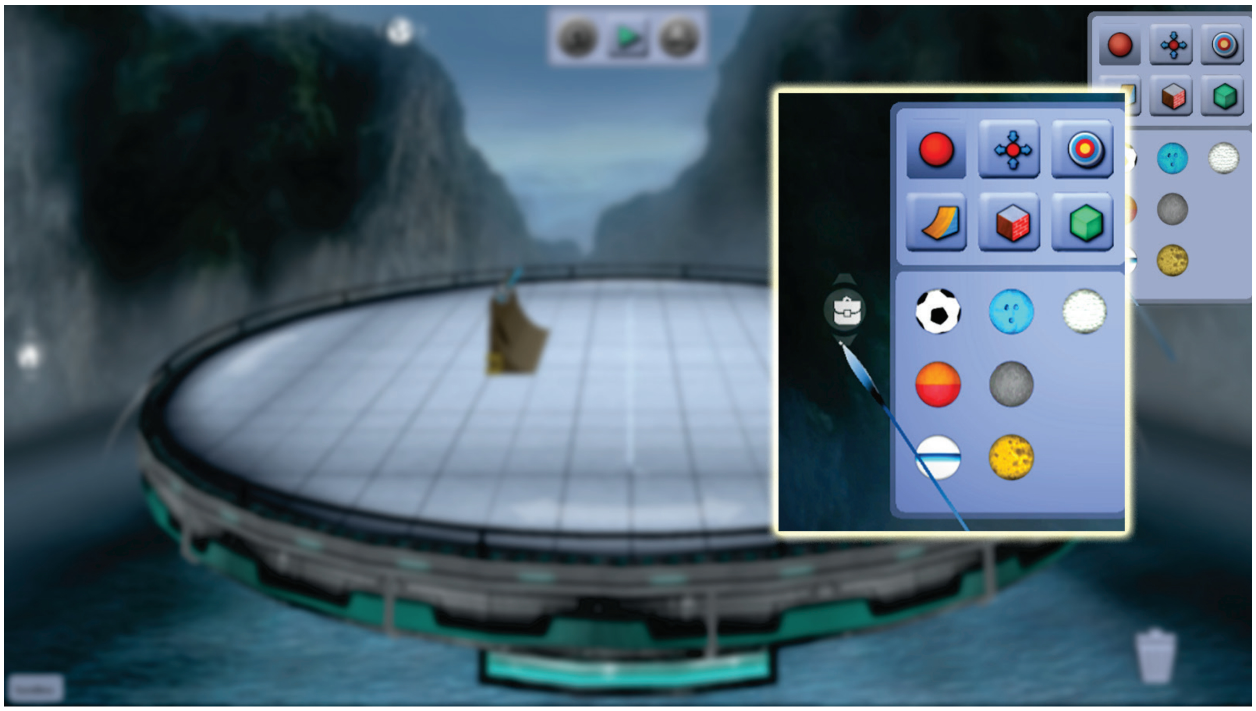
Tool Palette

A tool palette contains stylus tools, or modes, such as selection, scale, and the camera path tool. Some tools affect the stylus visualization directly as a mode, while other tools may appear as new objects, held and temporarily dropped into the scene or on models.

This tool palette from Studio shows a ruler, clipping plane, and camera.

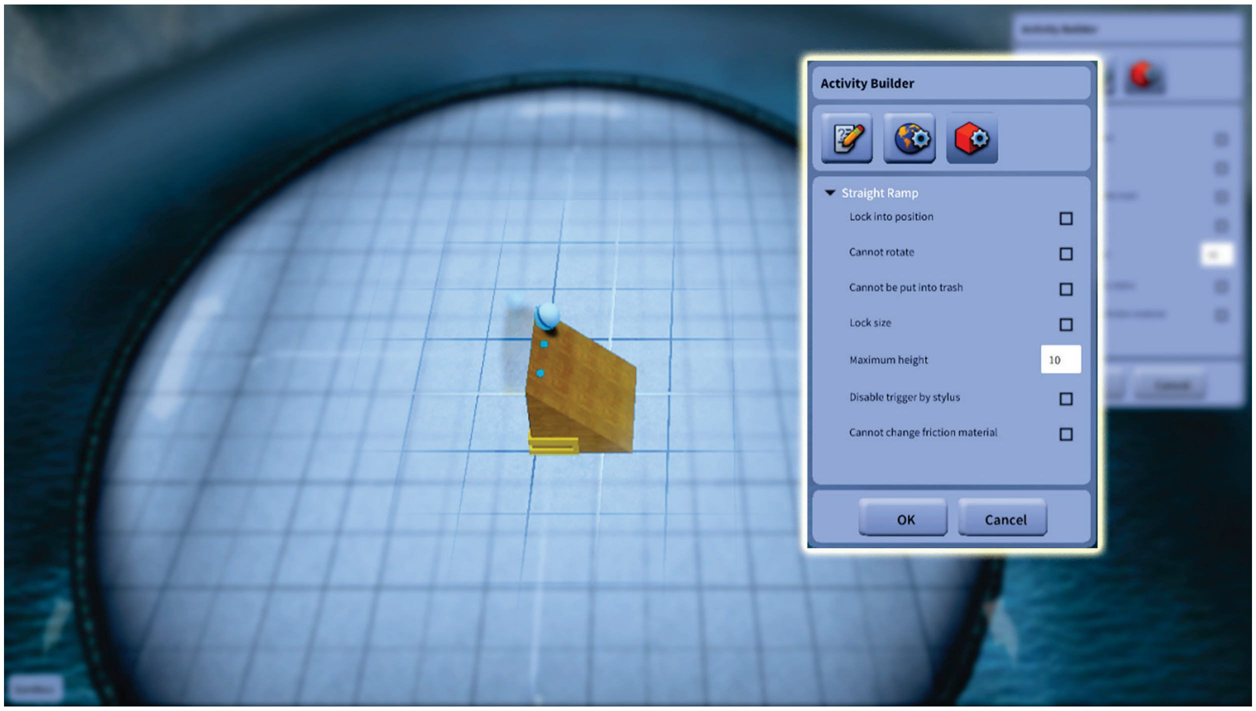
Backpack

The *backpack* (or *inventory*) provides quick access to available objects, or recently imported 3D models, that the user may add to their scene. Backpack objects may be brought out and into the scene by simple drag-and-drop using the stylus. If low detail models are available, they are shown both in the backpack UI as well as during the drag-and-drop operation; otherwise, a placeholder of the model is shown until dropped in the scene. Some backpack items may have specific quantity limits enforced (for the given lesson and learning objectives), while others may be unlimited. Some inventories can also be added to by drag-and-drop from the scene back into the backpack.


Backpack shown in Newton's Park

Inspector

The inspector shows the details of an object. The inspector updates in response to new user selections in the scene, showing the properties of the most recently selected object.

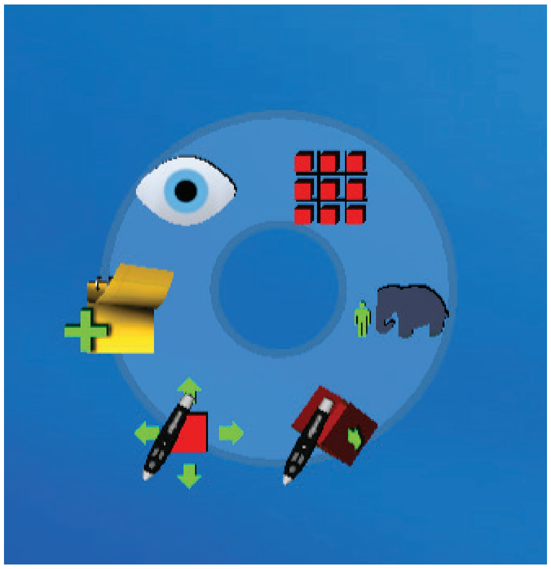


Straight Ramp
- Lock into position ☐
- Cannot rotate ☐
- Cannot be put into trash ☐
- Lock size ☐
- Maximum height 10
- Disable trigger by stylus ☐
- Cannot change friction material ☐

Inspector shown in Newton's Park

Context Menu

zSpace has integrated context menus into our STEM applications. Here is an example from zSpace Studio.



The context menu in zSpace STEM applications is made up of a ringed disc and 3D icons. When activated, the ring appears around the stylus beam and remains present in the scene until the user clicks an icon on the menu or clicks away. Context menus containing different icons can appear when intersecting the scene vs. a model vs. UI elements. It is helpful for the icons on the context menu to have tooltips appear when intersected by the stylus.

A context menu can be placed in different parallax levels for a user's convenience. In zSpace applications, the scene-based context menu appears near zero parallax, making it easy to find, read, and use. The object-based context menu appears in front of the intersected object, regardless of parallax level, making it clear what object is being manipulated. The object-based context menu must never clip through the object itself, or it could become unusable.

The context menu always billboards towards the user. If a context menu would appear fully or partially off-screen, based on where the stylus was pointing when the menu was activated, the context menu moves to the center of the display, near zero parallax.

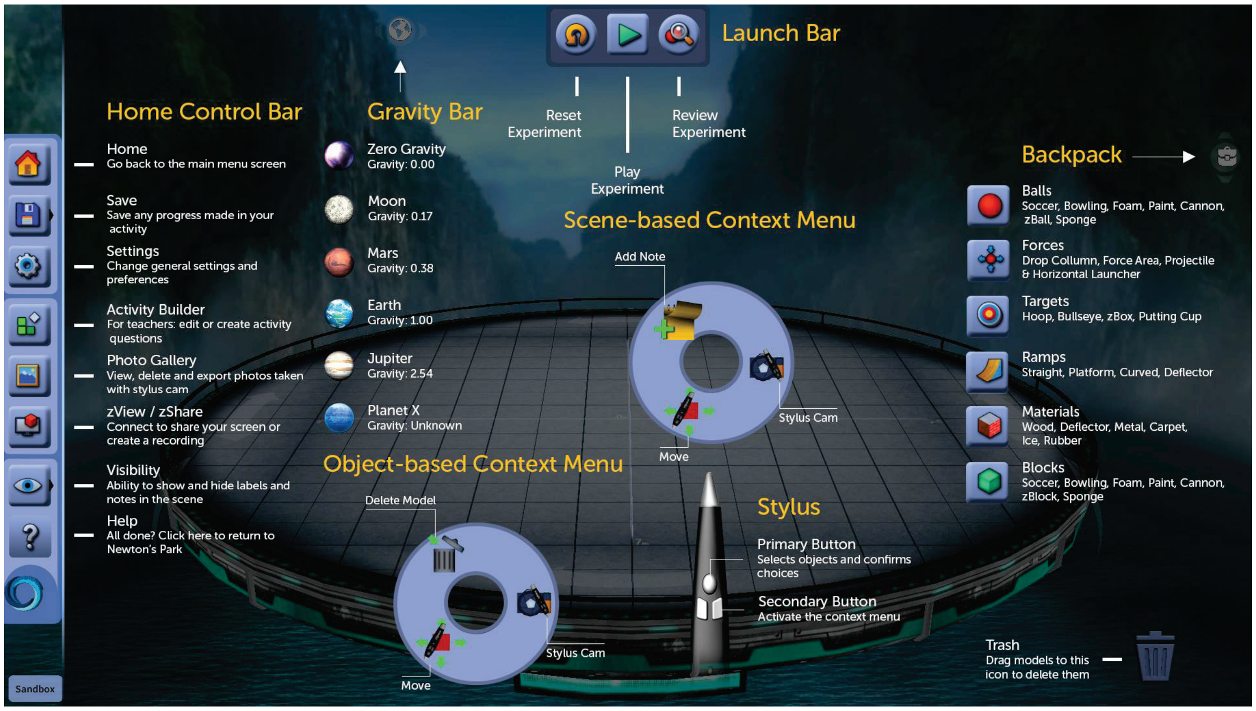
Trash

Trash is represented by the trash can icon in the lower corner of the screen. To delete objects from the scene, the user may drag-and-drop the object into the trash. When the stylus intersects with the trash, the user can drop the held object to delete it. Feedback is both visual and aural, with animation to different states (lid opens and closes), and sounds when objects are deleted. The icon itself is somewhat flattened in appearance and placed at zero parallax, making it easier to access and distinguish as distinct from the central scene.

Help Screen

Help screens are available at various places in the application, revealed by clicking the ‘?’ button in the control bar, or pressing F1 on the keyboard. All help screens feature:

- Short descriptions of functionality, with arrows and lines connecting to where this functionality may be found.
- An image of the stylus, with short descriptions of each button mapping.
- An image of one or more expanded context menus, with descriptions of each choice.



Home Control Bar

Home
Go back to the main menu screen

Save
Save any progress made in your activity

Settings
Change general settings and preferences

Activity Builder
For teachers: edit or create activity questions

Photo Gallery
View, delete and export photos taken with stylus cam

zView / zShare
Connect to share your screen or create a recording

Visibility
Ability to show and hide labels and notes in the scene

Help
All done? Click here to return to Newton's Park

Gravity Bar

Zero Gravity
Gravity: 0.00

Moon
Gravity: 0.17

Mars
Gravity: 0.38

Earth
Gravity: 1.00

Jupiter
Gravity: 2.54

Planet X
Gravity: Unknown

Scene-based Context Menu

Add Note

Stylus Cam

Move

Object-based Context Menu

Delete Model

Stylus Cam

Move

Stylus

Primary Button
Selects objects and confirms choices

Secondary Button
Activate the context menu

Backpack

Balls
Soccer, Bowling, Foam, Paint, Cannon, zBall, Sponge

Forces
Drop Collumn, Force Area, Projectile & Horizontal Launcher

Targets
Hoop, Bullseye, zBox, Putting Cup

Ramps
Straight, Platform, Curved, Deflector

Materials
Wood, Deflector, Metal, Carpet, Ice, Rubber

Blocks
Soccer, Bowling, Foam, Paint, Cannon, zBlock, Sponge

Trash
Drag models to this icon to delete them

Sample Help Screen

STEM Content Creation

You can create STEM content using zSpace Studio and Unity 3D or zSpace activity builder.

Studio Models

zSpace Studio uses the Unity 3D engine for rendering. For more information about authoring assets, including models, animated models, and activities, please refer to the following sources:

- zSpace Support Studio: How to. This page includes a link to the *3D Model File Importing Guide* and other resources.
- zSpace Studio Publishing Guide (please contact your zSpace representative).

STEM Activities

- zSpace Activity Authoring edu.zspace.com/info/activity-builder

Thanks for reading!