ShadowLight: an immersive environment for rapid prototyping and design

Kalev H. Leetaru*
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

ABSTRACT

ShadowLight is a virtual reality application that provides an immersive environment for multipurpose design and evaluation. Unlike traditional design tools that provide a built-in set of manipulators keyed to a particular set of design tasks, or evaluative systems that provide limited manipulation capabilities, ShadowLight offers a loosely defined environment that is capable of supporting the unique needs of both audiences. ShadowLight provides an atmosphere that is flexible enough to support rapid prototyping and design tasks, while at the same time permitting a richness of extensibility that allows scientific and industrial tasks to be performed using the same environment. ShadowLight defines only a basic interface that is extended through the development of plugins. The collection of plugins that is loaded at any given moment defines the capabilities available, and hence what the application "becomes" to its user. ShadowLight specifically seeks to address the current state-of-the-art in which several different environments must be employed for design, evaluation, and exploration tasks. In their place, ShadowLight attempts to provide a single environment that can be selectively extended to address the unique needs of a particular application field.

Keywords: virtual prototyping, rapid prototyping, virtual reality, immersive design, immersive systems, virtual reality toolkits, virtual environments

1. INTRODUCTION

Humans reason about the world around them and react to it in terms of what they are able to perceive. The process of manifesting an idea into tangible form is therefore inherently spatially oriented. However, conventional media do not provide an intuitive environment for the expression and refinement of spatially oriented design. Traditional computer modeling software systems such as Maya provide a three-dimensional medium, but partially negate the effectiveness of that medium by constraining the user with a two-dimensional interface. Physical models provide a three-dimensional interface, but often impede the refinement process by making modifications more difficult.

Virtual prototyping applications seek to address these issues and provide a more natural environment for design by exploiting immersive virtual reality. Industrial tools such as VisualEyes provide turnkey environments that often incorporate domain-specific evaluation and simulation components. However, most of these tools permit very limited interactive manipulation in the immersive environment, encumbering their effectiveness in the actual design phase of development. The domain-specific nature of their evaluation components also restricts their use in other fields. Artistic tools such as Surface Drawing provide a general-purpose spatial sketching environment more suited to the schematic design phase where interactive creation and iterative design are crucial. However, the output of these tools must then be imported into other environments to continue the evaluation stage of design.

Virtual prototyping systems are thus often stratified into domain-specific evaluation tools and freeform sketching tools. Domain-specific tools, with their limited manipulation facilities, target the production stage of design.

Further author information -
K.H.L (correspondence): Email: leetaru@ncsa.uiuc.edu; Project Web site: http://shadowlight.ncsa.uiuc.edu/
evaluation, with its greater demands on fidelity and analysis. Sketching tools, on the other hand, support the initial schematic phase, with its emphasis on dexterity and immediacy.

ShadowLight attempts to address the unique needs of both audiences, providing a loosely defined environment capable of sustaining interactive schematic design using a variety of media and supporting the multifaceted evaluation of that design. Rather than providing a built-in set of manipulators keyed to a particular set of design tasks, ShadowLight instead defines only a basic set of interfaces against which plugin authors develop. These plugins are self-contained applications, with their own interaction, processing, and simulation logic. ShadowLight dispatches user interaction events to plugins, but each plugin is responsible for taking its own actions based on that input. This allows for a very diverse spectrum of possible plugins, ranging from simple drawing tools to complete embedded simulations. A freeform ribbon or tube plugin may have a very simple click-and-drag interface to draw a new shape in space, while a sculpting tool may have significantly more complex interface needs. A physical simulation, such as a fluid flow, might have any number of input and interactive control parameters. In each case, the plugin developer is entirely free to leverage devices ranging from floating palettes and popup menus to proprietary widgets and gestural interfaces. The result of each plugin is a set of objects added to the space. In the case of sketching or sculpting tools, these objects may be static polygonal meshes, while in the case of more advanced tools, the result could be dynamic bodies with highly evolved behaviors.

The ability of each plugin to provide its own interaction and simulation logic allows plugin applications as diverse as simple brush strokes to reactive animated objects and complex physical simulations as the media for creation and evaluation. A given design created in ShadowLight may consist of interactively drawn static brush strokes and polygonal elements side-by-side with interactively placed intelligent agents and physical simulations. ShadowLight provides a consistent and intuitive interface to this functionality, seamlessly integrating the differing media into a single design environment.

ShadowLight thus provides an environment that is flexible enough to support rapid prototyping and design tasks, while at the same time permitting a richness of extensibility that allows scientific and industrial tasks to be performed using the same environment. The collection of plugins that is loaded at any given moment defines the capabilities available, and hence what ShadowLight “becomes” to its user. An architect might primarily utilize the sketching tools, while a researcher or an instructor might focus on the physical simulation tools. However, at any time, the architect might choose to augment a design with a fluid simulation to recreate a pool, while an instructor might use a sketching tool to trace out the oscillation of a wave.

ShadowLight specifically seeks to address the current state-of-the-art in which several different environments must be employed for design, evaluation, and exploration tasks. In their place, ShadowLight seeks to provide a single environment that can be selectively extended to address the unique needs of a particular application field.

2. THREE GENRES OF VIRTUAL REALITY DESIGN

Currently there are three primary genres of software that address the actual design phase of creation in virtual reality: sculpture, drawing, and constraint-based. Applications in each genre offer a palette of tools specifically targeted to that single modality. Sculpting and drawing applications seek to extend traditional techniques by replacing the artist’s slab of clay or paintbrush dipped in paint with a virtual counterpart that transcends the inherent restrictions of the physical world. Constraint-based applications similarly venture to augment traditional Computer Aided Design philosophy, with its existing reliance on virtual tools, to incorporate the immersion and spatial freedom of virtual reality.

If the focus is on artistic, rather than construction-oriented design, then the flowing, almost instinctual, forms of organic sketches are preferable to the predefined primitives and snap-to-grid doctrine of constraint-based design. In freehand drawing and sculpting applications, freedom is precisely the focus, and the lack of facilities for geometric shapes is not a restriction. Tools such as 3DM, HoloSketch, 3-Draw, and CavePainting all provide differing sets of drawing manipulators. CavePainting, for example, provides a subset of a painter’s brushes, enhancing them
by allowing geometrically and materialistically complex strokes. BLUI \(^8\) and others \(^9\) provide an analogy to the sculpting environment, by providing a volume in which “blobs” may be added and subtracted to form complex shapes. However, the moment the focus shifts to structural design, often heavily dependent on geometric constructions, organic representation becomes a significant hindrance to manifestation. Constructing a perfectly flat wall by sketching in three dimensions is an enormously intricate and time-consuming task.

Constraint-based applications such as VADeT \(^10\) seek to address the unique needs of geometric design, but do so by sacrificing much of the spatial freedom of organic design and instead limit their collections of tools to construction-centric manipulators. VADeT, for example, limits the user to a library of predefined primitives and textures. The focus is on assembly, rather than the multifaceted conundrum of creation.

3. COEXISTENCE-OF-SPACES VERSUS OBJECT-BASED SPACE PARTITIONING

Rather than restricting itself to a limited palette of sketching, sculpting, or constraint-based instruments, ShadowLight permits tools from all three genres to coexist in a single design environment. A user can sketch an organic brush stroke right beside a polygonal mesh or constraint-defined object. Thus, perhaps the closest correlates to ShadowLight’s flexibility are the virtual reality “operating systems” such as eNai. \(^11\) These systems allow multiple VR applications to be run together in a single viewing environment. The programs execute entirely independently of each other, and user input is channeled to the application with the current “focus”. The driving force of these systems is the subdivision of the active space into discrete regions dedicated to each application. ShadowLight supports this model, but extends it by allowing plugins to define their interaction boundaries. For most plugins, these boundaries are defined on a per-object basis, such as an individual ribbon or sculpture, while certain objects such as polygonal meshes, may even be shared between plugins. ShadowLight’s emphasis is therefore on the creation, deposition and perturbation of objects in space, rather than the coexistence of spaces, encouraging significantly finer-grain usage of the design space.

An example of the benefit of object-based space partitioning over the coexistence-of-spaces model may be found in the use of one plugin to annotate or visually extend another. For example, a meteorologist might have a plugin that displays a severe storm simulation and allows its various parameters to be modified and examined. To note a specific region of interest in the storm, he or she might wish to use a freeform sketch plugin to highlight and annotate that region.

In the coexistence-of-spaces model, these two plugins would act as independent applications, each given a region of space dedicated to its operation. Every user action performed in a plugin’s region of space is relayed to that plugin for processing. The simulation plugin, for example, might treat one axis of its space as a timeline and any clicks in that space cause the simulation to be relocated along that timeline, while the sketching plugin might treat any click as a request to begin drawing a new freeform object. From the standpoint of a coexistence-of-spaces model, this presents no problems, as each plugin receives notification only of user actions performed within its region of space. This requires, however, that the regions of space be disjoint, as any overlap in the spaces would cause ambiguity over which plugin should receive notification of user events occurring in that overlapping region. Hence, under this model, there would be no way for the researcher to use the freeform sketch tool to apply annotations directly on top of the storm simulation. Under ShadowLight’s object-spaced space partitioning model, however, the visual output of the storm simulation would be considered simply an object in space, and the user would be free to use the sketching tool to annotate the space around it.

4. HARDWARE INDEPENDENCE: OBJECT-BASED INTERFACES

One of the specific goals of the coexistence-of-spaces model is to allow unmodified applications to coexist in the same virtual workspace. The requirement of disjoint spaces is merely a technical byproduct of this goal that allows the applications to coexist and yet not be aware of each other. The object-based model, however, enforces a specific interface metaphor that developers must adhere to in order to gain the additional flexibility and expressive power of
an object-based environment. This can sometimes require interface modifications to existing applications that are ported to this model.

In the example simulation plugin described above, one axis of the simulation plugin’s region of space is used as a virtual timeline, with any clicks along this axis being translated into a repositioning of the simulation along its timeline. If the application was originally developed for a CAVE™ or other fixed-axis virtual reality facility, this interface may follow naturally from the physical dimensions of the facility. In the object-based environment of ShadowLight, however, there is no longer a fixed axis along which clicks can be measured. Thus the developer will be forced to modify the application to use a different interface modality, such as a visible palette-like timeline that floats in space near the simulation.

The interface modality that ShadowLight enforces through its object-based model is not necessarily a limitation. In fact, it actually benefits application developers by implicitly forcing them to abstract their interfaces away from assumptions about the underlying physical hardware. Interfaces such as the example timeline interface rely on a fixed, easily-discernable, axis being defined by the physical display facility. This type of interface is most commonly used in a CAVE, where the horizontal distance between the side walls defines a fixed axis along which this measurement may be calculated. However, in a head mounted display or curved-wall system, the endpoints of the axis no longer have a well-defined physical correspondence and the system must define an arbitrary set of endpoints. Without the physical correspondence, however, the user is now burdened with locating these endpoints without visual assistance. By forcing the developer to use an interface metaphor abstracted from the physical hardware, such potential interface problems are avoided.

5. VR DEVELOPMENT LIBRARIES AND PLUGIN MODELS

ShadowLight may be thought of both as an end-user application for virtual prototyping and design and as a development system for designing new virtual reality applications. In the former, ShadowLight provides a runtime environment defined by the set of currently-loaded plugins. In the latter, ShadowLight provides a rich framework and supporting API that allows developers to write applications that can be used for various design and evaluative tasks, all the while coexisting in a single environment.

Most virtual reality applications are developed using low-level libraries such as CAVELib, FreeVR, or VR Juggler, that provide an abstraction from the underlying hardware and offer basic utility functions. The developer is left to implement all graphics, interaction, and other logic required of his or her application. The majority of applications are therefore stand-alone highly-specialized applications that perform a narrow set of specific tasks. The added interface and implementation complexities of the VR environment make it even more difficult to extend applications or combine them together, discouraging code reuse and collaboration. ShadowLight attempts to address these issues through its plugin architecture, allowing applications to leverage a preexisting API of high-level functions for faster development and enabling independent applications to execute side-by-side and have their output spatially combined.

Most virtual reality operating systems provide an immutable kernel that handles low-level tasks and use their runtime extensibility to execute applications side-by-side. Other runtime extensible VR toolkits use their runtime extensibility to allow many internal aspects of the toolkit to be extended, including its kernel. In the case of VirtualExplorer, a framework forming the skeleton of a generic virtual reality application is provided to be extended by the developer by writing plugins to override specific functionality such as rendering, IO, or user interaction handling. The application core is also written as a plugin or series of plugins that are inserted into the framework. Bamboo takes this idea a step further and defines no core framework other than a kernel that provides a plugin harness: program logic at all levels from disk IO to application-specific functionality are provided through plugins.

While this approach has the benefit of providing tremendous flexibility, it forces the developer to be aware of all aspects of the toolkit’s operation and even a simple extension may require modifications on multiple levels. DIVE
17 attempts to provide some of this flexibility while alleviating the complexities of internal toolkit modifications by using a scenegraph as its “kernel”. Applications are written as a set of plugins that manage objects in this scenegraph and dispatch and process various events triggered in the scenegraph. The plugin architecture defined by DIVE revolves around the manipulation of this scenegraph and a complex event hierarchy that defines the interaction of objects within it.

The DIVE approach has many benefits, but requires that a unified description “language” exist that can describe all possible objects in the world in a semantically-rich way. The expressive power of this language must be sufficiently great to enable DIVE’s complex event hierarchy to be built on top of it. The requirement that all objects exist in the scenegraph in a semantically rich form can cause difficulties when dealing with partially-defined objects. For example, a simulation plugin has been written in ShadowLight that connects to a remote simulation resource and accepts a stream of commands to update a visual representation of that simulation in the ShadowLight environment. The user is able to interact with the simulation, adjusting select parameters that are sent back to the simulation resource. In this case, the plugin’s internal data structures are only able to represent the information it has about the object, which is its current visual representation and the values of the selected parameters that can be interactively adjusted. The rest of the simulation parameters and the data sources that define the simulation are available only to the remote simulation resource: in essence the plugin is acting as a “dumb client” to the simulator. In the absence of the true “definition” of the simulated object, the DIVE model begins to break down, as it can only store a collection of polygons in the scenegraph with no useful semantic description.

6. THE EVOLUTION OF SHADOWLIGHT

ShadowLight began as a polygon-based sketching tool that allowed the rapid construction of simple worlds in virtual reality. The author had experienced firsthand the complexities of designing worlds on the desktop, importing them into VR, compiling a list of desired changes, and then making those changes to the model back on the desktop. Rather than iteratively designing his virtual worlds on the desktop and previewing them in virtual reality, he wanted a solution that would allow him to design and evaluate the worlds entirely in virtual reality. Existing tools did not support the rich interface metaphors that he desired and so he developed the first version of ShadowLight.

As ShadowLight found its first user community in the University of Illinois School Of Architecture, the limitations of its original interface became apparent. The first version of ShadowLight had attempted to bring a traditional desktop modeling interface into three dimensions and had thus brought with it the notion of a mode-based control panel, as seen in Figure 1. While the architects using ShadowLight were comfortable with mode-based interfaces on their desktop software, they found its VR counterpart to be extremely encumbering. Informal user studies suggested the context-aware interface seen in Figure 2. This interface proved so successful that its use was continued in the most recent version of ShadowLight, seen in Figure 3.

![Figure 1](image1.png) Mode-based palette from ShadowLight 1.
As its user community grew, ShadowLight was pulled in several different directions. Architects wanted a complete continuum of manipulation and control over their designs, while school children wanted a simplistic “rubber-stamp” creation ability, and researchers wanted the ability to employ remote tools and evaluative processes within the design environment. It was realized almost immediately that the best way to address these needs was to add a plugin capability to ShadowLight to allow these differing user communities to employ their tools within a single application framework.

The largest group of users has been a senior/graduate-level university architectural design studio, which was taught five consecutive semesters using ShadowLight in a CAVE facility. On average most of the students became relatively fluent with the application within twenty to thirty minutes after their first experience. Students became comfortable enough with the application over time to develop their own workflows and construction techniques within the environment. For example, while the first two versions of ShadowLight did not support curved surfaces, students developed their own techniques for approximating them using the available geometric shapes. Although most of the students had previous CAD or other desktop modeling experience, the students cited the unique immersive capabilities of the VR environment as exciting and empowering for spatially evaluating their designs in full-scale.

An educational outreach program for seventh- and eighth-grade middle school girls was also conducted using ShadowLight. This three-term project sought to present high technology in an alternative interface to the traditional mouse and keyboard stereotype. Despite having no previous experience with virtual reality, the girls in this program demonstrated significant fluency with the interface, exhibiting a fair degree of confidence in their actions after just a few sessions.

The current version of ShadowLight builds on two previous generations of the underlying infrastructure and interface, spanning four years of continual use. This ongoing interaction with users from varied backgrounds has provided a rich source of guidance on interface and feature improvements.

The new extensibility of ShadowLight has led to its recent adoption as an application virtual prototyping environment. Rather than using ShadowLight to prototype designs, some developers are now using its plugin framework to write collections of special-purpose plugins that can be reused and combined with other plugins. In one example, several existing VR applications are being ported to ShadowLight plugins so that they can be used together in a single evaluation environment.
7. DESIGN: THE SHADOWLIGHT APPROACH

Design in ShadowLight is based around the notion of a “world”, an infinitely bounded space that serves as the medium in which the designer composes. To interact with this world, a six-degree of freedom “wand” is used which provides three buttons and a joystick. The wand is tracked over a certain physical space in front of a stereo-projected surface. The user may move about the world by pointing the wand in the desired direction of travel and pushing forward on the joystick. No global collision detection is performed, so the user is free to travel to any location in the world through any path, to freely explore the space from locations not possible in real life.

Creation in ShadowLight is based around the available selection of plugins, which are enabled through a popup menu that appears when the user depresses the right wand button in space. This popup menu is called the global menu and allows the user to select which plugin to use as the current “brush”. The currently selected plugin receives all subsequent “unattached” user interaction events. Each click of a button or movement of the wand or joystick is processed by ShadowLight to see if the user is selecting or interacting with an existing object or performing an action in unoccupied space. If the action is in unoccupied space, it is forwarded to the active plugin to process.

As an example of this, a basic sketching tool has been developed that allows simple freeform strips to be drawn in space. To use this tool, the user right-clicks in an unoccupied region of space and selects the “Ribbon” tool from the global menu as seen in Figure 3. Then by depressing and holding the left wand button and moving the wand through space, a flat strip tracing the wand’s movement is left behind. This tool is similar to the paintbrush props of CavePainting and is a useful preliminary sketching and annotation tool, as seen in Figure 4.

![Figure 4 Sketching using the Ribbon plugin.](image)

The richness of the ShadowLight plugin environment is demonstrated by a fluid simulation plugin that has been developed. This plugin provides the ability to insert simple reactive fluid bodies into a world, where they can be freely placed and inter-located with other static and interactive elements. An architect, for example, might use this plugin to more accurately reproduce the experience of an atrium design that includes a reflecting pool. The ability to physically co-locate the fluid bodies produced by this plugin with other objects in the world, as seen in Figure 5, lends a greater degree of flexibility than environments that require strict spatial separation. This flexibility to combine objects in the same physical space allows the architect to insert a fluid body into the atrium design and then use additional modeling plugins to generate the surrounding container geometry. An instructor can use the same fluid plugin to teach an introduction to the wave properties of fluids, by creating several fluid bodies with differing properties and then inviting students to interact with the fluids in a hands-on fashion.
This fluid simulation defines a rubberstamp interface for depositing instances of the simulation at given locations. Work is underway to modify the simulation to use grab points to resize the water surface and a floating palette to define its simulation characteristics. Using this approach, a researcher could deposit multiple independent copies of a simulation side-by-side in a single world with differing initial conditions, and evaluate them simultaneously.

The variety of media that can be created by plugins in ShadowLight makes it impossible to generate a unified view of a given world’s objects. It can therefore be difficult to create evaluative plugins that operate upon other objects, such as a virtual wind tunnel plugin that simulates air flow around a given design. Without semantic knowledge of the objects in a scene, these plugins don’t have the information they need to accurately carry out their simulations. ShadowLight’s current approach to this issue is to require such evaluative plugins to either include their own selection of design tools that generate objects that can be evaluated by that plugin, or to have third-party “certified compatible” plugins that have been designed to work with that plugin. The designer is still free to use all available plugins in creating the world, but only those design tools that are compatible with a given evaluation plugin may be evaluated by it.

8. DEVELOPMENT: THE SHADOWLIGHT APPROACH

The flexibility that ShadowLight affords users of its design environment does not permit the semantically-rich unified scenegraph approach of toolkits like DIVE. Objects in ShadowLight may have vastly different representative needs and thus ShadowLight takes the approach of having each plugin act as an independent application, managing all object-centric data structures internally. ShadowLight mediates between the plugins by directing user events and other key resources, while plugins interact with the user by updating a shared scenegraph. Unlike the DIVE approach, this scenegraph is used only for geometry storage, providing the visual representation of the world. Plugins add, update, and remove objects from the scenegraph to provide visual feedback to the user, and request notification of user events via the ShadowLight API to react to user actions.

In a way, the ShadowLight kernel acts very much like a virtual reality operating system, managing a set of unrelated plugins and allowing them to execute independently in a single display and interaction environment. However, ShadowLight takes this metaphor a step further by using an object-based model and requiring plugins to add collections of objects to the scenegraph rather than encapsulating sets of raw OpenGL calls.

Plugins in ShadowLight use a set of API calls to manage the polygonal representations of their objects in the scenegraph. Much like a traditional scenegraph, collections of polygons are encapsulated in a hierarchy of nodes, each of which is uniquely addressable by the plugin that created it. Although all objects are managed in a single shared scenegraph, each plugin has a “view” on the scenegraph that allows it to see only its own objects. Using a shared scenegraph allows ShadowLight to perform global optimizations such as transparency sorting, while the
independence of views allows plugins to manage the scenegraph without trampling each other. To allow developers to create suites of plugins that are designed to interact with each other, ShadowLight does make available API calls that allow plugins to access nodes created by other plugins, but this is not the general behavior.

Most plugins developed for ShadowLight are special-purpose tools that add objects, such as brush strokes or simulations, to a world. These plugins are either designed specifically as ShadowLight plugins, or are ported from existing virtual reality applications. As part of a contract for the United States Army Corps of Engineers, a plugin was developed for ShadowLight that allowed the rapid integration of remote resources into the ShadowLight environment. The plugin allows a remote application to issue a stream of commands to create, manage, and destroy objects in a ShadowLight world. As described earlier, plugins in ShadowLight act as independent applications, interacting with ShadowLight only to update the scenegraph and receive user events. Hence, a plugin that allows a remote application to update the ShadowLight scenegraph enables an entire new class of virtual reality application development. Rather than specifically designing or redesigning an application to work in a virtual environment, this “remote control interface” can be used to rapidly “virtual reality enable” an application. An application displaying vector fields, for example, could simply send those vector lines as objects to ShadowLight via this interface rather than being entirely rewritten to work in a virtual reality facility.

9. CONCLUSION AND FUTURE WORK

Traditional design mediums impose significant encumbrances on representation and evaluation. Virtual prototyping tools offer an escape from these restrictions, but constrain the user to either domain-specific evaluation environments or single-task sketching spaces. VR operating systems allow multiple independent applications to coexist in a single space, but are based around evaluation and segmented interaction rather than full-space design. Existing VR plugin frameworks require significant effort to develop applications within them, or enforce significant encumbrances like unified description languages. ShadowLight attempts to address these limitations by providing a flexible and loosely defined medium in which many different design modalities can coexist. This medium embraces the entire design cycle by allowing the designer to combine organic and artistic instruments in the initial schematic drawings, while permitting a seamless transition to later stages of development.

This leads us to ask, What is ShadowLight? The answer to that question depends on who is using it. To an architect, it is a virtual prototyping tool, while to a middle school student, it is an educational platform, and to a scientist it is a tool for rapid comparison of simulations. In short, ShadowLight becomes a different tool to different people, providing an environment in which an idea may be intuitively explored, evaluated, and refined, using whatever combination of representations makes the most sense.

Although ShadowLight has been used for more than four years, the plugin-enabled version is a very new technology and as such does not yet have many plugins developed for it. Work is ongoing to develop a full complement of general-purpose sketching tools. We are also collaborating with another university in developing a set of educational tools that will benefit from the ShadowLight environment.

ACKNOWLEDGEMENTS

Parts of this project were funded by the US Army Corps of Engineers Construction Engineering Research Laboratory under contracts DACA42-01-D-0004-03 and DACA42-01-D-0004-007. The support of the National Center for Supercomputing Applications for the use of its CAVE facility as well as the expertise and support of its staff are gratefully acknowledged, along with the various user communities who have provided valuable feedback regarding the ShadowLight environment. The author would especially like to thank Alan Craig for his mentorship on this project.
REFERENCES