Interactive Manipulation of 3D Scene Projections

Category: Research

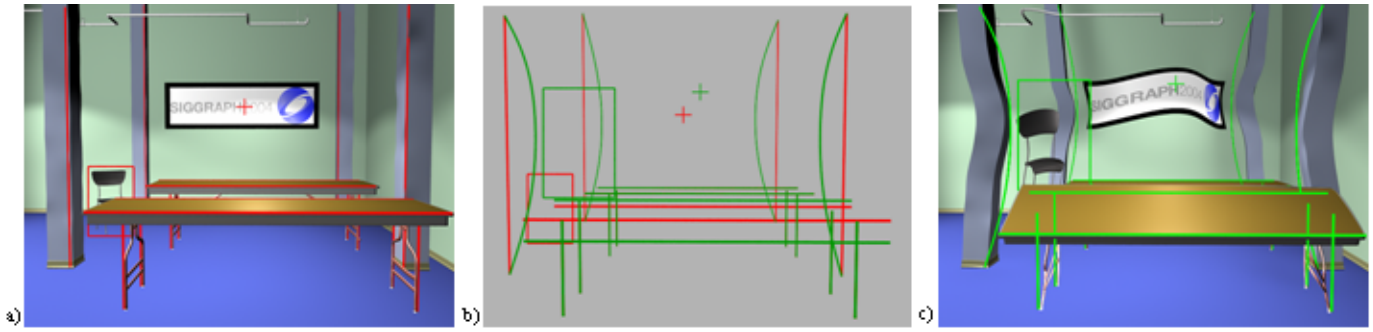

Figure 1: Defining a nonlinear projection of a 3D scene. a) Original scene from a default view showing sketched 3D features. b) Manipulating the 2D projections of the 3D features. c) The new, nonlinear projection. d) Thumbnails of the individual cameras used to construct the image. e) Geometry colored by the camera weights.

Abstract

Linear perspective is a good approximation to the format in which the human visual system conveys 3D scene information to the brain. Artists expressing 3D scenes, however, create nonlinear projections that balance their linear perspective view of a scene with elements of aesthetic style, layout and relative importance of scene objects. Manipulating the many parameters of a linear perspective camera to achieve a desired view is not easy. Controlling and combining multiple such cameras to specify a nonlinear projection is an even more cumbersome task. This paper presents a direct interface, where an artist manipulates in 2D the desired projection of a few features of the 3D scene. The features represent a rich set of constraints which define the overall projection of the 3D scene. Desirable properties of local linear perspective and global scene coherence drive a heuristic algorithm that attempts to interactively satisfy the sketched constraints as a weight-averaged projection of a minimal set of linear perspective cameras. This paper shows that 2D feature constraints are a direct and effective approach to control both the 2D layout of scene objects and the conceptually complex, high dimensional parameter space of nonlinear scene projection. The simplicity of our interface also makes it an appealing alternative to standard through-the-lens and widget based techniques to control a single linear perspective camera.

CR Categories: I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling

Keywords: Camera control, Projection, Sketch interface, Perspective, Rendering, Non-linear perspective

1 Introduction

Projection, together with occlusion in a 2D image, provide a viewer with information about the 3D spatial relationship between scene objects. Loosely speaking, projection relates objects in a left-right, top-bottom sense orthogonal to the viewing direction, while occlusion gives us depth ordering. Linear perspective, besides being a good approximation to the human visual system, is able to provide us with depth information in 2D without the need for occlusion. Artists expressing 3D scenes balance linear perspective with elements of style and the relative importance of scene objects to create projections that are both informative and aesthetically pleasing.

1.1 Motivation

Creating informative and appealing 2D projections of 3D scenes is a challenging task not only in Computer Graphics but in traditional artistic media as well. For staged 3D scenes, the composition of scene objects is intertwined with its flattening to a 2D projection. Photographers, for example, explore many camera settings and viewpoints while rearranging scene objects to achieve a desired shot. Artists, on the other hand, mentally rearrange and stitch together different views of a scene to render a desired 2D projection onto canvas. Even within the bounds of realism, artists very often deviate from strict linear perspective for two chief reasons.

First, linear perspective can be too restrictive to display all of the necessary parts of a scene. It also may not be able to meet all of the desired 2D framing and layout constraints. While this can be fixed somewhat by rearranging objects in a staged scene, often that is not a feasible option. Second, viewers are not overly critical of deviations from a linear perspective as long as the overall projection has a notion of global scene coherence and a locally continuous projection (preferably linear perspective) [**?**].

Artists also introduce deliberate distortions of perspective for a number of reasons (see Figure 2). Perspective distortions can be used to convey a mood, vary the relative importance of objects, and simulate wider view angles as seen with fish-eye lenses and eye secades. Singh [Singh 2002] provides a range of motivational images by artists Escher, Hockney and Picasso that show the deliberate use of nonlinear projection in art.

Figure 2: Non-linear projection in art (*Jean Fouquet, circa 1480*).

Current camera control techniques offer the user direct control over the various parameters of a linear perspective camera. Controlling these parameters to obtain a desired projection of even a single object can be difficult, particularly when the object is off the center of the viewing axis. The global nature in which linear perspective camera parameters affect the projection can make the specification of a desired projection for multiple scene objects almost intractable. While there has been some recent work on the specification of nonlinear projections constructed from multiple linear perspectives [Agrawala et al. 2000; Singh 2002; Grimm 2001], these systems similarly require the users to achieve the desired 2D projection indirectly, by controlling and combining a large number of camera parameters.

In this paper we thus explore a direct interface to camera control for scene projection.

1.2 Approach

We start out with a small number (typically one) of default viewpoints (see Figure 1a), that are defined using a conventional Computer Graphics camera. The artist now sketches simple 3D geometric proxies, such as points, lines, and boxes, directly into the scene. These geometric proxies reflect primitive shapes that artists traditionally use to lay out a 3D scene on a canvas. The 3D geometric proxies are then projected into 2D and become the visual handles through which the artist manipulates the projection. The artist then interactively re-sketches or manipulates the 2D features to specify changes to the current projection of the 3D features (see Figure 1b).

The altered projections of the 3D geometric proxies onto the 2D canvas become constraints that, along with other desirable projection properties, interactively define an overall projection of the 3D scene. This projection attempts to maintain the property of local linear perspective as much as possible, while introducing nonlinearities when necessary to satisfy the given constraints.

We represent the nonlinear projection as a weighted average of multiple linear perspectives as proposed by Singh [Singh 2002]. This is a very large solution space, consisting of all of the individual camera parameters plus their weightings. Completely constraining the solution would be a daunting task. Instead, we use knowledge of the artist's intent, plus the default viewpoint, to reduce the problem space.

Each of the 2D features represents a desired deviation from the corresponding default view. Our features are designed so that changes to them map naturally to perceived changes in the projection. To demonstrate the flexibility of the feature set we show how an arbitrary set of features, plus the default view, can be combined to control a single, linear perspective camera. The solution to this smaller problem is found using a nonlinear constraint satisfaction algorithm, which takes as input the features and the allowable changes to the default view, and returns a set of camera parameters which best meets these constraints.

We build upon this single camera solver to define nonlinear projections using multiple cameras. We essentially look for the smallest set of single cameras that satisfies the 2D features and has the property of local linear perspective. The algorithm attempts to group features that are proximal in image space and fit each group with a single linear perspective camera. Once the group of cameras is determined, we define weighting functions based on the corresponding 3D features.

There are a small number of nonlinear projections, for example panoramas and fish-eye views, that arise often enough to warrant a specific feature of their own. Each of these features creates two or more cameras which are then incorporated into the multiple camera algorithm as a unit.

1.3 Contributions

This paper contributes the first direct interface to controlling the 2D layout and projection of a 3D scene. Specifically, the contributions of this interface are threefold. First, we describe a rich set of feature primitives that constrain different parameters of a linear perspective camera. These features can be used individually or in combination to form a compelling widget set for interactive linear perspective camera control. Unlike previous constraint-based approaches [Gleicher and Witkin 1992] we define a family of explicit behaviour sets for the under-constrained case. Second, we describe a novel algorithm for the construction of continuous nonlinear projections from sketched feature primitives, with properties of global scene coherence and local linear perspective. Thirdly, we define complex feature primitives that are specifically designed to constrain popular nonlinear projections such as panoramas and fish-eyes.

1.4 Overview

The paper is organized as follows. Section 2 positions this paper relative to previous research in nonlinear projection, camera control and sketch interfaces. Section 3 elaborates on the system interface and workflow. Section 4 defines our feature set and discusses how changes to the 2D features are reflected in the allowable camera changes. Section 5 describes our approach to finding a single linear perspective camera that satisfies the 2D feature changes. In Section 6 we extend the single camera solution to multiple cameras, define weight functions, and how to interpolate between cameras. Section 7 describes the design of complex features to capture certain popular nonlinear projections. Section 8 provides the conclusion.

2 Related work

Image-space constraints have been used to control camera animations [Blinn 1988; Gleicher and Witkin 1992], automatic camera control for teleconferencing-type applications [Drucker and Zeltzer 1995], and automatic composition [Tomlinson et al. 2000; Gooch et al. 2001]. Gleicher [Gleicher and Witkin 1992], Blinn [Blinn 1988], and Tomlinson [Tomlinson et al. 2000] all used image-space constraints and a general purpose solver to create a camera animation. All of these approaches used only seven of the eleven camera parameters (position, orientation, and focal length), a small vocabulary of image constraints (point constraints, size in image, and a notion of "up") and a simple heuristic for any unconstrained parameters (keep them the same as before). We extend image-space con-

straints to all eleven parameters, introduce a comprehensive family of image constraints, and, most importantly, introduce a set of heuristics for managing the unconstrained parameters. Because changes in the 2D projection of 3D geometry can be accounted for by a variety of camera parameter changes, a naive implementation of a constraint system can result in (from the user's point of view) very unexpected behavior. Our heuristics constrain the remaining degrees of freedom by taking into account *how* the user is manipulating the 2D projections; this results in a much more stable, but not restrictive, system.

There are a variety of ways to solve for the camera parameters given a set of constraints. Gleicher [Gleicher and Witkin 1992] introduced a version that uses a standard Inverse Kinematics solution to solve for the *change* in camera parameters that will reduce the constraint error. The system iterates until it converges. The advantage of this approach is that it reduces to a least-squares problem for which solutions are well-understood; the disadvantage is that it requires a fair amount of machinery to create new constraints and these constraints must be expressible as quadratic constraints on the camera parameters or image points. We have experimented with both this type of gradient-based solver and the simplex solver and have found them roughly equivalent in terms of convergence speed and robustness. It is much simpler, however, to introduce and combine new constraints using the simplex solver.

Image warping techniques[Fu et al. 1999; Dorsey et al. 1991; Zorin and Barr 1995; Seitz and Dyer 1996] are inherently two-dimensional approaches with limited ability to explore different viewpoints. View-dependent distortions to three-dimensional scene geometry for animation and illustration[Martin et al. 2000; Rademacher and Bishop 1998] are geared more towards animating deforming characters and less towards scene composition. Multi-perspective panoramas capture three-dimensional camera paths into a single image [Wood et al. 1997; Rademacher 1999; Peleg et al. 2000]. These approaches provide little control over varying the importance and placement of different objects in a scene and are also not well suited to interactive manipulation.

Both Agrawala et al. [Agrawala et al. 2000] and Grimm [Grimm 2001] present a multi-projection approach where each object in the scene is assigned to some camera and rendered based on the linear perspective of that camera. The multiple renderings are composited to generate the final image using a visibility ordering of the objects from some master camera view. This approach provides good results for multiple discrete projections but does not handle projections continuously varying over objects as seen in Figure 5 or Figure 1. Agrawala also encapsulates a small number of camera manipulations (dolly-in plus zoom, fixed view, fixed position, and orientation) to make it easier for the user to create certain effects. Our system also supports these operations through the perspective, orientation, and size constraints.

The idea of constructing a nonlinear projection as a combination of multiple linear perspectives was presented by Singh[2002]. As presented, Singh's approach does not integrate well into a conventional animation workflow or have ways to control global scene coherence. We, on the other hand, use the notion of a default perspective cameras in conjunction with scene constraints to induce distortions of geometry such that the view from within a conventional linear perspective camera appears nonlinearly projected.

3 System Interface and Workflow

The user-centric workflow in this paper ultimately drives the underlying framework for defining the overall scene projection. In the physical world an artist has their own view of the 3D scene and a 2D canvas upon which to render it. Similarly, the artist's view of the 3D scene is captured in our setting by a conventional linear perspective camera which we call the exploratory view (see Figure 1a)

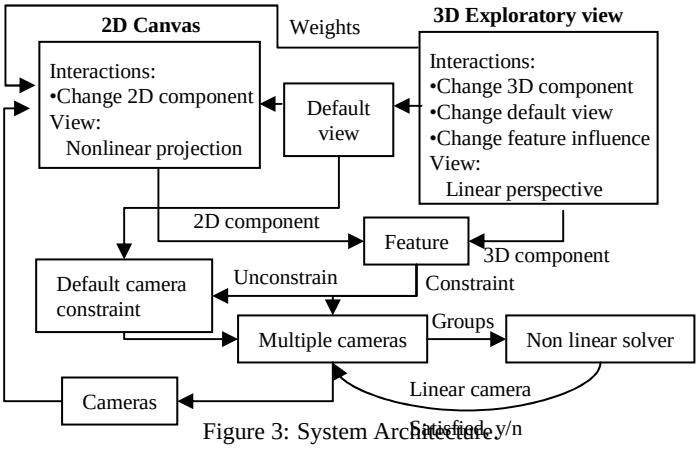

Figure 3: System Architecture.

and a 2D canvas (see Figure 1b). The 2D canvas allows interactive manipulation of the projected 3D constraints and is where the modified 3D scene projection appears (see Figure 1c). The exploratory view can be controlled using current techniques or those described in this paper, and is used to define the default camera.

In the physical world an artist first picks one or more viewpoints with which to render parts of the scene. While viewing the scene from one of these viewpoints, the artist begins sketching a few key features of the scene. This sketch defines the overall layout and projection of the scene in 2D. The sketched features are placed to balance the perceived view(s) of the scene with artistic intent.

Similarly, within our framework the artist first finds one or more viewpoints of interest by directly manipulating the exploratory view. They next define, by sketching, 3D geometric features of interest. These are points, lines, boxes or curves that are placed on objects (or their bounding boxes) in the 3D scene. These 3D proxies correspond to the set of features artists use to simplify the geometry of the scene. Now we bring to bear some of the advantages of working in the digital realm. The 3D geometric proxies are automatically projected based on the artist's viewpoint of interest. The artist can then edit these features, leave them unchanged, or override them entirely by sketching them afresh. The artist can delete existing scene features to unconstrain the scene projection or add new ones by sketching in 3D.

Unlike a traditional artist who has to fill the details of the overall scene projection after sketching the features, in our system the overall scene is always projected onto the canvas and interactively changes to meet the constraints imposed by the features sketched by the artist. Figure 3 shows the elements of user interaction as part of the overall system architecture.

A typical user interaction session is as follows. As the user manipulates the exploratory view it is projected simultaneously on the 2D canvas. The user can bookmark a view and label any view as a default view. 3D features can be interactively added, removed, and edited in the exploratory view. Each sketched feature is matched with a default view; typically the geometry is sketched in the corresponding default view, but it doesn't have to be. This allows the user to introduce constraints on features that are not visible in the default view.

As the user creates 3D features, their projections appear in the 2D canvas. The user can then edit the projected features in the 2D canvas, and our system will interactively compute a new projection to match the requested changes, while maintaining, as appropriate, fidelity to the corresponding default view.

At some point the user will introduce constraints that cannot be satisfied with a single linear camera. At this stage the can toggle between viewing the scene non-linearly with all constraints enabled, or continue to use linear projection with a user-selected subset of

the constraints enabled (the system shows the disabled constraints in their manipulated, not projected, locations). Non-linear rendering is, in general, not real-time; the locally linear mode allows the user to fine-tune the linear projection in one area, while using the sketched 2D proxies to indicate what is happening in the remainder of the scene.

The user can also use the exploratory view to change the 3D drop-off functions associated with each of the 3D features. These drop-off functions determine the contribution of each local linear camera to the overall scene projection at arbitrary points in space.

Although our system supports multiple default views, for simplicity most of this paper assumes a single, default view.

4 Feature Primitives

We now describe the feature primitives that a user manipulates to control the overall projection.

From an artistic standpoint the perspective projection of an object can be thought of as having four components: Position, size, orientation and perspective. The first two relate to the location and size of the object on the canvas. The next two define the viewing transform relative to the center of the object. Orientation can be broken into three components, rotation in the image plane ("spin"), left-right rotation, and up-down rotation. Perspective is related to the view distance from the object and is artistically conceptualized by defining vanishing points for families of parallel lines [o'Connor Jr. et al. 1998]. While there may be a large number of linear projections that satisfy a given feature constraint, the type of feature and how it has been changed suggests an order of preference of the feature's control over the four different components.

As an example, the simplest feature primitive is a point. Figure 4a shows the default view of a scene with a specified point on the table. We intuitively expect the camera to pan (translate perpendicular to the viewing axis) when the point is moved elsewhere on the canvas, rather than changing the size, orientation, or perspective. Similarly, if we have a line in the scene, and the line has only been translated, we expect only the position component of the camera to change. If, however, we rotated the line about its center, we would expect the orientation to change. Scaling the line would produce a size change. Details of the implementation can be found in 5.2.

Our set of feature primitives are based on three principles.

- The feature primitive shapes are simple and traditionally used by artists to lay out scene projections.
- The feature set provides successive coverage of the four components of linear perspective.
- Some feature primitives that control different components of linear perspective combine effectively to define new, more comprehensive primitives.

4.1 Anatomy of a Feature

Each feature has a 3D component and a corresponding 2D component. Where appropriate, the 2D component has handles for moving the entire feature, rotating it in the image plane, and scaling it (either uniformly or in a single direction). Depending on how the 2D feature has been altered, we unconstrain the corresponding camera component. Each feature generates a constraint (Section 5) which is sent to the constraint solver (Section 6) along with which camera components are unconstrained. The system determines, for the entire set of features, which camera components are still constrained; each component generates a constraint which is also sent to the solver (Section 5.2).

4.2 Features

We now describe our set of feature primitives:

Point: The simplest feature primitive. The point feature unconstrains the position component of the camera. If more than one point feature is present the size, orientation, and perspective components are unconstrained, in that order.

Line: A line is essentially two point features, but the *relationship* between the points is important. The line unconstrains position, orientation in the plane, and/or size, depending on the how the line has changed (see Figure 4d).

Wedge: A wedge is built from three points and is useful for specifying the position, full orientation, and size. Changing the position, size, and orientation of the wedge unconstrain the corresponding camera components. Changing the angle of the wedge and the relative lengths of the edges changes the left-right and up-down orientation of the object. A variation of this feature is the **Orientation wedge**, in which the position of the wedge is ignored.

Two lines: This feature is built from four points, usually arranged as two parallel lines, although this is not required. This feature behaves as the line does for affine transformations. Full orientation and perspective are unconstrained when the relative angles and sizes of the lines change. This constraint is motivated by the definition of vanishing points in art and is thus useful with objects that have natural parallel or orthogonal features, such as a table. The **Vanishing point** is a restricted form of this constraint, defined using two parallel lines, that only controls perspective.

Cube edge: This constraint is built from six point constraints and represents the edge of a cube. The user positions a cube around the object and chooses which edge to manipulate. This feature is the most general, and can be used to control all of the camera parameters. Again, affine transformations of the feature act to unconstrain position, size, and image space orientation, while changing the relative angles completely unconstrains the camera.

Bounding box: The 3D component of this feature is one or more objects (or their bounding boxes). The 2D component is a box. The position and size components of the camera are changed so that the 3D component projects inside of the box. A variation of this is the **size box** which only constrains the projected size of the object.

Rotation: The 3D component of this feature is a set of orthogonal axes; the 2D component is the projection of those axes. This feature strictly controls the orientation of the object and allows a default view orientation to be defined for an object. Often, objects like a table or chair only define a partial default orientation in terms of an **up** vector feature.

We describe two complementary combinations of the above features that can be used to completely specify a linear projection.

Bounding box-Rotation: These two features completely control the size, position, and orientation of the object, but not the perspective (the constraint solver will use the perspective of the default view).

Bounding box-Orientation wedge-Vanishing-point: These three constraints together control all four projection components.

In addition to the projection constraints themselves, the user can indicate if they want to allow oblique projections, non-uniform scaling, or very narrow and wide projections.

5 Constraining a Single Camera

Once the user has specified one or more features, the system must find the camera that best meets those feature constraints. This problem is very closely related to the one of camera calibration in computer vision [Zhang 2000]. In computer vision, the problem is usually stated as follows. Given a set of 2D locations for known 3D features (usually points, lines and sometimes conics), find the extrinsic (position and orientation) and intrinsic (focal length, aspect ratio,

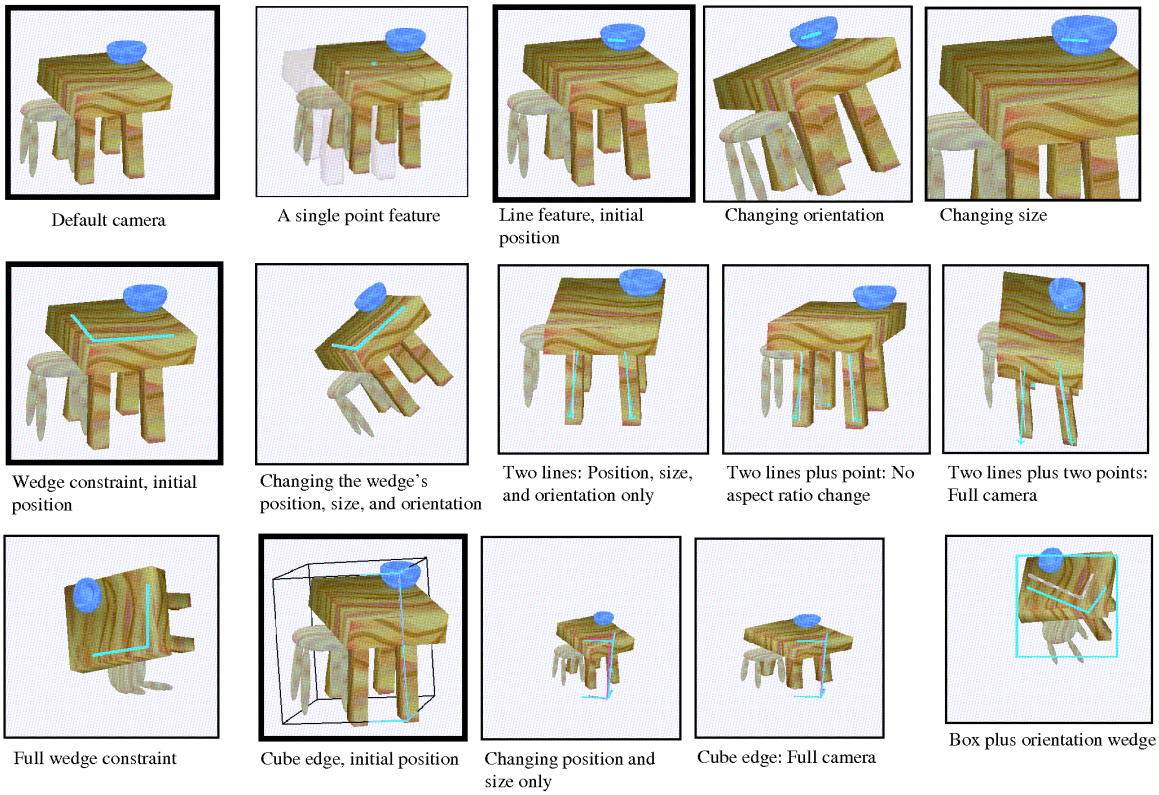

Figure 4: A taxonomy of feature primitives.

skew, center of projection) camera parameters that map the 3D features to their 2D image locations. Unfortunately, for a perspective camera with 3D features in general position there is no closed-form, linear solution. Additionally, there are 11 camera parameters (6 extrinsic and 5 intrinsic) so there must be at least as many independent constraints as there are camera parameters to fully specify the camera. We restrict the parameter search space from this general setting with feature primitive definitions that carry knowledge of the user's intended change in projection from the default view. For example, we can assume the default view's center of projection and skew unless the user draws a feature that is explicitly designed to control these parameters.

Our basic approach is to use a general-purpose, non-linear solver to satisfy the feature constraints. Each feature f produces a constraint in the form of an error equation E_f to be minimized. Each constrained component of the camera produces another error equation, E_d. Each error equation is weighted then summed to produce the complete error equation:

$$E = \sum_d w_d E_d + \sum_f w_f E_f$$

The system provides default weights which can be over-ridden by the user if desired. The solver searches over the space of camera parameters to find the set that minimizes this equation.

We use the standard method for building a perspective matrix [Foley et al. 1990] from a rotation (3dof), translation (3dof), focal length (1dof), aspect ratio (1dof), center of projection (2dof), and skew(1dof). Each feature has access to its 3D components, 2D components, the default camera D, and the camera currently under consideration C. Some notation:

- $C(P) = p$ is the projection of a 3D point P into a 2D screen point p by the camera C. This encapsulates the matrix multiplication, the homogeneous point normalization, dropping the depth component, and scaling to the width and height of the screen. $C(\vec{V}) = \vec{v}$ will be the projection of the vector $\vec{V} = P - Q$, found by taking $C(P) - C(Q)$.
- We will use upper case for 3D points, and lower case for 2D points. The subscript d indicates the desired screen-space location of the 2D feature.

5.1 Feature Constraint Equations

For each feature in Section 4 we need an equation that measures how well the constraint is satisfied. Note that the magnitude of the constraints is normalized to correspond roughly to pixel error, *i.e.*, if a point projects one pixel away from its desired location then the error function returns one.

Point: This equation measures the distance between $p = C(P)$, the projected 3D feature point, and the desired location p_d:

$$E_p = \|p - p_d\|$$

Line: Let P and Q be the end points of the line. We measure the difference in the projected end points.

$$E_l = (\|p - p_d\| + \|q - q_d\|)/2$$

Wedge: The wedge constraint is the sum of two line constraints, each scaled by half.

Orientation wedge: This constraint uses only the angle between the projected line and the sketched line, and the lengths. Scaling by 360 equates a one degree error with one pixel.

$$E_{ow} = \frac{1}{4}\big(360\,|\cos^{-1}(C(P0\vec{P}1)\cdot p0_d\vec{p}1_d)|/(2\pi) + |\|p0_d - p1_d\| - \|C(P0\vec{P}1)\|\| + 360|\cos^{-1}(C(P1\vec{P}2)\cdot p1_d\vec{p}2_d)|/(2\pi) + |\|p1_d - p2_d\| - \|C(P1\vec{P}2)\|\|\big)$$

Two lines: We could use four point constraints in this case, but we have found that the constraint captures the preference of camera parameters better if we include the difference in directions and lengths of each line pair as well. This prevents slight inconsistencies in the end point locations from dramatically changing the perspective. If $P0$ and $P1$ are the end points of one line then the equation for the first line is:

$$E_{tl1} = \frac{1}{4}\big(\|(P0+P1)/2 - (p0_d + p1_d)/2\| + 360|\cos^{-1}(C(P0\vec{P}1)\cdot p0_d\vec{p}1_d)|/(2\pi) + |\|C(P1\vec{P}0)\| - \|p1_d - q1_d\|\|\big)$$

and similarly for the second line.

Cube edge: This constraint is implemented as the sum of six point constraints.

Rotation: The rotation constraint specifies a desired orientation for a coordinate frame e_1, e_2, e_3 centered in the middle of the object. If we take just the rotation matrix of the camera and multiply it by the *x*, *y*, and *z* axes, we get the coordinate frame E_1, E_2, E_3. We measure the difference in these coordinate frames by taking the individual dot products:

$$E_{rot} = 360/3\sum_{i=1}^{3}(1-(e_i\cdot E_i))^2$$

The **Up** constraint is a special case of the rotation constraint, where we only consider e_2 (the *y* axis).

Bounding box: The bounding box equation measures the difference between a bounding box that contains the projected vertices v of the object, and the desired bounding box. Let p be the center of the projected vertices, *i.e.*, $p = 1/n\sum_n C(v)$, and w,h be the size of the box containing the projected vertices. Let p_d, w_d, h_d be the desired center and size of the 2D bounding box.

$$E_b = 1/3(\|p-p_d\| + |w-w_d| + |h-h_d|)$$

If aspect ratio changes are not allowed, then we constrain only one of the width or the height, whichever is less.

Size box: The size box equation is similar to the bounding box, except we measure the differences in the width and height only and ignore the center.

5.2 Default view constraint

In this section we define the default camera error function, E_d, for each of the camera components. Which equations are active depends on which camera components are constrained, whether or not the user is allowing center-of-projection and skew changes, and aspect ratio changes. To determine which camera components are unconstrained we initially set all components to be constrained, then walk through the current list of features and unconstrain the components the feature is editing.

For the following discussion, let C be the camera under consideration, and D be the default camera.

The following is a general-purpose way of measuring how much an input value v varies from a desired value $v_d \neq 0$. This equation equally punishes smaller and larger deviations and has a maximum magnitude of one:

$$E_r(v, v_d) = \begin{cases} 1-|v/v_d| & |v/v_d| < 1 \\ 1-|v_d/v| & |v_d/v| \geq 1 \end{cases} \quad (1)$$

Center of projection: The default value for the center of projection is $(0,0)$. To measure the deviation we simply measure the size of the current camera's center of projection u, v:

$$E_{cop} = 2|u|/W + 2|v|/H$$

where W, H is the size of the screen. If we wish to constrain the center of projection to a different value we use Equation 1. Note that changing the center of projection by 1 corresponds (roughly) to moving the center of projection to the top of the screen.

Skew: We use Equation 1, scaled by $(W+H)/2$ to measure the skew. The default value for skew is 1.

Position: Translation and center of projection are the primary parameters that influence position, but if an object is off the viewing axis then rotation will have an effect as well. Taking any point P on the optical axis of the default camera:

$$E_{pos} = \|C(P) - D(P)\|$$

which is essentially a point constraint. This will allow the camera to rotate around the optical axis.

Size: The focal length and translation along the view direction are the primary parameters influencing size. Usually, we prefer to change size by changing the focal length, leaving the translation, which also affects perspective, unchanged. Let $P_c =$ `eye + look`, $P_u =$ `eye + look` + (H/f) `up`, and $P_r =$ `eye + look` + (H/f) `right` be the points at the center, and center-top, and center-right of the film plane of the default camera (f is the focal length). Then the projected sizes of these vectors should be the same:

$$E_S = 1/2(E_r(\|C(P_u - P_c)\|, \|D(P_u - P_c)\| + E_r(\|C(P_r - P_c)\|, \|D(P_r - P_c)\|)$$

To measure the aspect ratio we use the ratio of the two film-plane vectors:

$$E_A = E_r(\|C(P_u - P_c)\|/\|C(P_r - P_c)\|, \|D(P_u - P_c)\|/\|D(P_r - P_c)\|)$$

Orientation: The parameters primarily influencing orientation are the rotations. The skew parameter can also influence the orientation, although in general allowing this parameter to vary results in unexpected camera views. Measuring orientation differences is identical to the rotation constraint described above. Note that if we use the "up", "look", and "right" vectors of the camera that we can separate orientation into rotation in the image plane, left-right rotation, and up-down rotation.

Perspective: Perspective depends primarily on the orientation and position of the object relative to the camera, although changing the center of projection and skew also play a role. We measure the change in perspective distortion by measuring the angle change of the edges of a cube placed a unit distance along the `look` vector.

$$\begin{aligned} P_{t/b} &= \texttt{eye} + \texttt{look} \\ P_{c_i} &= P_{t/b} + \texttt{look} + \pm\texttt{right} \\ E_P &= \frac{360}{8\pi}\sum_i E_r(tan^{-1}C(P_{c_i} - P_{t/b}), tan^{-1}D(P_{c_i} - P_{t/b})) \end{aligned}$$

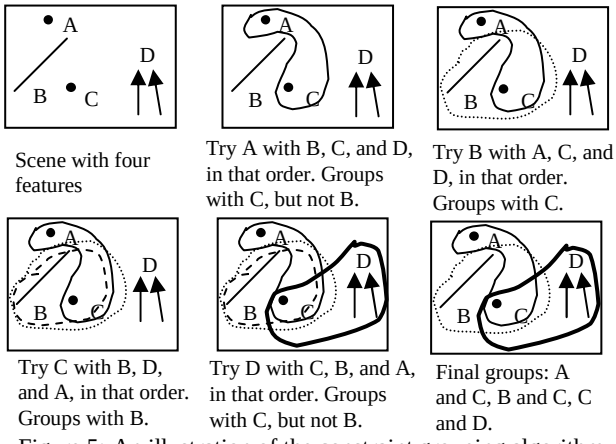

Figure 5: An illustration of the constraint grouping algorithm.

5.3 Weights

The final optimization equation is a weighted sum of all active constraint equations plus the default camera equations. We use the weights in two ways. First, changing the weight of an individual feature causes that feature to become "stronger". This provides the user with additional control in the case where not all constraints are satisfied. Second, changing all of the weights for the default camera constraints simultaneously allows the user to indicate how important the default camera is. The initial values for all feature constraints is one, while the default camera constraint weights are 100 because we want these to over-ride the feature constraints.

5.4 The Solver

We use a simplex, or amoeba [Nelder and Mead 1965] solver. This solver deals well with large numbers of parameters, parameters that have unequal effects, and does not require explicit derivatives. The solver requires the number of parameters, an initial starting condition, and an equation to minimize. It searches through parameter space until the decrease in the error function falls below a given threshold.

The initial conditions are the current camera parameters. If the user has explicitly forbidden oblique projections we can leave the corresponding parameters out. For computational efficiency reasons, we usually solve for the best translation and rotation parameters, then use those parameter values to initialize a search over the remaining parameters.

Except for dramatic constraint changes, the solver usually iterates 100-400 times before stabilizing.

Weighted constraint solvers are notoriously difficult to manage, can be very sensitive to perturbations in the weights, and can suffer from local minima. We are insulated from many of these problems for several reasons. We have a user in the loop who can (indirectly) nudge and guide the solver by making small changes to the image-space constraints. It is also visually clear which constraints are not being met and what happens as constraints are changed. In the following section it will be important to be able to determine if the constraints were actually met; because we know a mapping from the magnitude of each error constraint to approximate pixel error, we can determine appropriate thresholds for success.

¿¿ Table here showing timing results.

¿¿ Comparison figure to Gleicher's stuff.

6 Constraining Multiple Cameras

We represent continuous nonlinear projections as a blend of multiple linear perspective cameras [Singh 2002]. Given a set of features we now need to compute a minimal set of linear cameras to fit the feature constraints and a corresponding set of weight functions that describe how much each camera contributes to the projection of any point in the 3D scene.

We do this as a two step process. In the first step we find groups of features such that all of the feature's constraints can be satisfied, within a specified tolerance, by a single linear perspective camera. We take the smallest group such that each constraint is covered by at least one camera. We use a heuristic to drive a greedy search that captures local linear perspective in the image by attempting to group features that are sketched close to each other on the 2D canvas. Once the set of cameras has been determined the 3D feature components in each camera's group is used to define the camera's contribution to the overall projection of points in space. More specifically, the weight function for any camera is a summed distance surface implicit function [Singh 2002] defined around each 3D feature component that the constraint satisfies (see Figure 6).

We now look at the grouping algorithm in detail.

6.1 Determining the Cameras

We take a greedy approach to determining the grouping of features which are satisfied by a single camera. The input to the algorithm is a set of n features $F_1 \ldots F_n$ and their corresponding desired 2D projections $p_{1\ldots n}$. We define a set of n possible groups $G_1 \ldots G_n$ (and corresponding cameras) as follows:

For $i = 1$ to n
  $G_i = \{F_i\}$
  Sort features $j \neq i$ by $\|p_i - p_j\|$
    for $k = min_j$ to max_j
      if satisfied($G_i \bigcup F_j$)
        $G_i = G_i \bigcup F_j$

We now cull any duplicate groups or any group which is a subset of another. We illustrate this algorithm in Figure 5.

While this algorithm does not guarantee a minimum number of cameras it is simple and robust and does a good job of capturing local linear perspective in the image by first satisfying constraints of features that are close to each other with the same linear perspective. By covering features with as many cameras as possible, we ensure better interpolation results.

¿¿ A comparison figure here, showing multiple vs. few cameras

6.2 Determining Weights

Once the set of output cameras $C_1 \ldots C_m$ have been defined we need to determine weight functions that control their relative contribution to the projection of all points in space.

It is important that the regions proximal to the various 3D feature components be projected by the cameras that satisfy that feature constraint. The influence of one camera on the overall projection then falls off spatially from its 3D feature components at a rate that reflects how local the camera's projection is. Implicit surface primitives lend themselves perfectly to capturing such weighting functions. Each 3D feature component P_j defines a distance-based implicit function in 3D, called f_j. We use two parameters, r_{in}, r_{out}, for each feature that determine the area influenced by P_j. We set f_j to be one at any distance $r < r_{in}$, and zero past r_{out}. Between r_{in} and r_{out} the function falls-off in a typical bell-shaped blend function g [1]

[1] $g(x) = (x^2 - 1)^2, x \in [0,1]$ is an example of such a function.

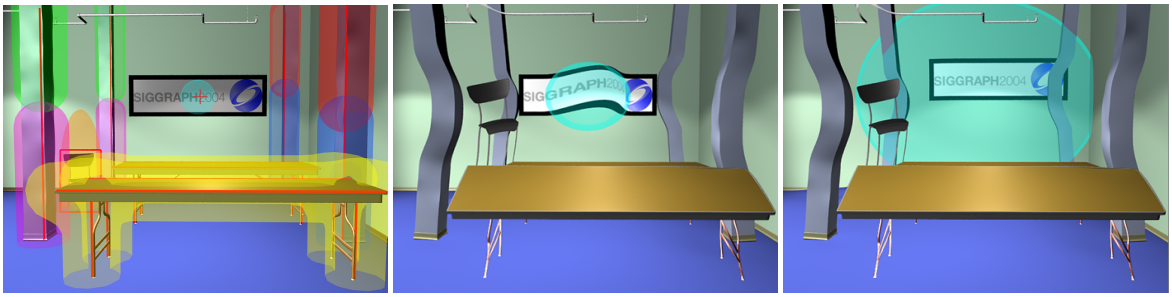

Figure 6: Implicit surfaces for the features in Figure 1. There are 8 cameras total (4 column cameras, 1 painting, 1 chair, 1 table and the default), and 14 features. Color indicates camera grouping. Middle and right: Changing the influence of the point feature in the painting.

$$f_j(Q) \quad = \quad \begin{cases} 1 & ||Q-P|| < r_{in} \\ g(\frac{\|Q-P\|-r_{in}}{r_{out}-r_{in}}) & r_{in} \leq \|Q-P\| \leq r_{out} \\ 0 & \|Q-P\| > r_{out} \end{cases}$$

For each output camera C_i, the weight function w_i is a simple summation of the weight functions of all of the features that are in the group G_i to which the camera C_i is fit. Given a point Q in the scene, we first calculate the weight for each camera. The weights are normalized if they sum to greater than 1. Points where the weights sum to less than 1 fall outside the locality of any of the fitted cameras. These points are blended in with their projection in the default camera view. The user can easily vary the influence different feature constraints have on the overall scene projection (see Figure 6) by controlling their drop-off distances directly in the exploratory view.

The nonlinear projection for any point in space Q is now computed as a simple blend of projections of the cameras $C_1 \ldots C_m$ and the default view D. The projection q of a point Q is now simply

$$q \quad = (1-\sum w_i)D(Q) + \sum w_i C_i(Q)$$

¿¿ Maybe a comparison figure here showing 3 different ways to merge cameras

7 Nonlinear Feature Primitives

Thus far our nonlinear projection algorithm takes a number of features, constructs groups that can be satisfied by a single perspective camera, and blends these cameras continuously. While the resulting nonlinear projection is likely to satisfy the specified constraints there is no mechanism for the user to define particular nonlinear projections. In keeping with our user-centric approach we would like the user to be able to specify specific types of nonlinear projections within our system framework. We accomplish this by defining complex feature primitives that must be satisfied by a pre-specified nonlinear projection. We demonstrate two feature primitives that correspond to a panorama view and a fish-eye projection.

¿¿ Better description of panorama

Panorama: A panorama can be generated by either spinning the camera in place or spinning it around an object. To specify a panorama we draw a line in the scene and a corresponding curve (which starts off as a line) in 2D. The center point of the curve can be moved to create an arc in the image. This arc is approximated by a small number of line constraints. We then generate one camera for each line constraint. We either constrain the camera to spin around its axis to spin around the center point of the arc.

The panorama feature is always grouped only with itself. The columns of Figure 1 are distorted using a panorama.

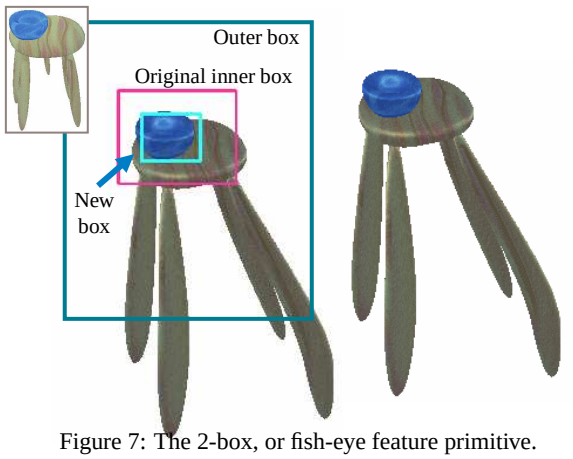

Figure 7: The 2-box, or fish-eye feature primitive.

2-Box or fish-eye: The 2-box feature is an extension of the bounding-box feature. It is defined using two concentric boxes. The outer box is treated as the usual bounding-box feature and participates in the specification of a nonlinear projection as described in Sections 3-5. Once the outer box has been satisfied by some camera C_{out}, we create an additional camera C_{in} that uses C_{out} as the default view. The camera C_{in} is allowed to translate along the viewing axis of the C_{out} camera, and if the center of the inner box has changed, perform a pan as well. The inner box is not included in the feature groups. The weight function for C_{in} is computed as a fall-off from the inner box to the outer box. As shown in Figure 7 the 2-box is an effective feature with which a user can specify and control a fish-eye or telescoping projection.

8 Conclusion

It is worth noting that a user can composite multiple 3D scenes or objects onto the same canvas in 2D without having to actually address their placement relative to each other in a common 3D scene. Each 3D scene would have its own exploratory view but the overall projection would be controlled and viewed on a single 2D canvas.

We have presented an interactive technique for specifying nonlinear projections. Our approach addresses the high-dimensional space problem by making a series of heuristic decisions based on expected image-space behavior. These heuristics are visually encapsulated in a rich set of 2D primitives. By using a non-linear solver and a default camera we can easily allow arbitrary combinations of feature primitives. We then combine features into coherent groups, using a set of heuristics based on desirable properties of projections. This combination results in a very flexible system that provides as much control as possible to the user while still allowing them to interactively control nonlinear projections.

¿¿ another example

References

AGRAWALA, M., ZORIN, D., AND MUNZNER, T. 2000. Artistic multiprojection rendering. In *Eurographics Rendering Workshop 2000*, Eurographics, 125–136.

BLINN, J. 1988. Where am i? what am i looking at? In *IEEE Computer Graphics and Applications*, vol. 22, 179–188.

DORSEY, J. O., SILLION, F. X., AND GREENBERG, D. P. 1991. Design and simulation of opera lighting and projection effects. In *Siggraph*, ACM Press, ACM, 41–50.

DRUCKER, S. M., AND ZELTZER, D. 1995. Camdroid: A system for implementing intelligent camera control. In *1995 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 139–144. ISBN 0-89791-736-7.

FOLEY, J., VAN DAM, A., FEINER, S., AND HUGHES, J. 1990. *Computer Graphics : Principles and Practice*. Addison Wesley.

FU, C.-W., WONG, T.-T., AND HENG, P.-A. 1999. Computing visibility for triangulated panoramas. In *Eurographics Rendering Workshop 1999*, Eurographics, 169–182.

GLEICHER, M., AND WITKIN, A. 1992. Through-the-lens camera control. *Siggraph 26*, 2 (July), 331–340. ISBN 0-201-51585-7. Held in Chicago, Illinois.

GOOCH, B., REINHARD, E., MOULDING, C., AND SHIRLEY, P. 2001. Artistic composition for image creation. *Eurographics Workshop on Rendering*.

GRIMM, C. 2001. Post-rendering composition for 3d scenes. *Eurographics short papers 20*, 3.

MARTIN, D., GARCIA, S., AND TORRES, J. C. 2000. Observer dependent deformations in illustration. In *NPAR*, ACM Press, ACM, 75–82.

NELDER, J. A., AND MEAD, R. 1965. A simplex method for function minimization. In *Computer Journal*, vol. 7, 308–313.

O'CONNOR JR., C., KIER, T., AND BURGHY, D. 1998. *Perspective Drawing and Application*. Prentice Hall.

PELEG, S., ROUSSO, B., RAV-ACHA, A., AND ZOMET, A. 2000. Mosaicing on adaptive manifolds. *IEEE Transactions on Pattern Analysis and Machine Learning 22*, 10, 1144–1154.

RADEMACHER, P., AND BISHOP, G. 1998. Multiple-center-of-projection images. In *Siggraph*, ACM Press, ACM, 199–206.

RADEMACHER, P. 1999. View-dependent geometry. In *Siggraph*, ACM Press/Addison-Wesley Publishing Co., ACM, 439–446.

SEITZ, S. M., AND DYER, C. R. 1996. View morphing: Synthesizing 3d metamorphoses using image transforms. In *Siggraph*, ACM Press, ACM, 21–30.

SINGH, K. 2002. A fresh perspective. In *Graphics Interface 2002*, 17–24.

TOMLINSON, B., BLUMBERG, B., AND NAIN, D. 2000. Expressive autonomous cinematography for interactive virtual environments. In *Proc. of the 4th International Conference on Autonomous Agents*, ACM Press, 317–324.

WOOD, D. N., FINKELSTEIN, A., HUGHES, J. F., THAYER, C. E., AND SALESIN, D. H. 1997. Multiperspective panoramas for cel animation. In *Siggraph*, ACM Press/Addison-Wesley Publishing Co., ACM, 243–250.

ZHANG, Z. 2000. A flexible new technique for camera calibration. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22.

ZORIN, D., AND BARR, A. H. 1995. Correction of geometric perceptual distortions in pictures. In *Siggraph*, ACM Press, ACM, 257–264.