Our goal was to create an aesthetically pleasing and mechanically elegant interactive user experience, using the isometric perspective. Our preliminary research focused on common computational and artistic techniques for isometric projection. This research has been applied in the development of an isometric, world-builder engine.

**Overview**

Isometric projection is a form of axonometric projection in which the angle between any two of the three axes is 120 degrees. In effect, this creates a three-dimensional drawing with no vanishing point. In order to project a 2D sprite into isometric space, its Cartesian coordinates \((x, y)\) must be manipulated in order to derive an isometric position (described as \(x, y, z\)), using the following formula:

\[
\text{isoX} = \text{CartX} - \text{CartY} \\
\text{isoY} = (\text{CartX} + \text{CartY}) / 2
\]

The above formula assumes a known isoZ coordinate. Once a coordinate has been projected, to move it up or down along the isoZ axis, you simply use the original CartY axis.

Another computational issue that must be dealt with is the order in which sprites are drawn. We used a technique called depth sorting, in which the isometric coordinates of each sprite are compared against those of the rest, and sprites with higher values are drawn on top of sprites with lower values.

*Figure 1.* An isometrically projected cube. As an axonometric projection, all lines on the same plane are parallel to each other. The 120° angle between any given two axes defines this projection as isometric specifically.

*Figure 2, Figure 3 (left to right).* Illustration of the effect of depth sorting on isometric assets. Both of these figures contain the same assets in the exact same configuration; the only difference is that the assets in the figure on the right have been depth sorted.
In addition to perspective drawing, this effect is reinforced artistically within our engine through the use of shading. Every asset is shaded to create the illusion of an upper right lightsource in the world, which contributes to the illusion of dimensionality in 2D sprites. We also chose an artstyle characterized by mellow colors and exaggerated details[Fig. 4], in order to avoid clashing with the inherent affectation of the isometric perspective.

**Challenges**

Because isometric sprites with “higher” coordinates obscure sprites with “lower” coordinates, we decided to implement a world-rotation mechanic to allow users to view all details of the world. This was achieved, computationally, by abstracting the world-layout data from the code for its visual representation. Once we could manipulate the appearance of the world without modifying the underlying data structure, we created four separate algorithms for reading and presenting the data, one for each rotation angle. To convince the user that they were seeing more than sprites simply being repositioned and resorted on a screen, each sprite sheet asset was also created with a separate frame to represent each rotation angle.

Another roadblock we faced was a psychological phenomenon known as multistable perception[2]. In our case, when the brain was not given proper visual context, it was possible to perceive some of our assets as either inset OR outset from their surroundings. To minimize this effect, we increased our assets’ height proportions relative to their widths and depths. This prevented the top of each asset from perfectly aligning with the top of the asset behind it, and helped convince the brain that it was a distinct, outset, form.

*Figure 5, Figure 6 (left to right). Normal height assets versus adjusted height assets. Due to the fact that the top sprite aligns perfectly with the sprite behind it, it can be difficult to interpret the figure as outset from its surroundings. The image on the right addresses this issue simply by increasing the height of the sprites so that the lines behind the top sprite are now slightly occluded.*
One of the most challenging problems we faced was allowing the user to interact in isometric worldspace using a mouse. The issue being that mouse coordinates are received as a simple cartesian, x-y position. While it is possible to project this point as an isometric point using the formula above, there is no way to know which isoZ plane with which the user is attempting to interact. Previous methods of dealing with this issue involve performing a collision check between the mouse cursor and an existing object in the scene, then using the object’s isoZ plane as the interaction plane. This places many limitations on the interaction, and is also rather confusing in terms of usability.

Our solution was to create an “interaction grid”[Fig. 7], comprised of isometrically projected, “flat”, tiles that are sorted along with the rest of the world’s assets. The user could then move this grid up or down the isoZ axis to specify exactly where they wanted to interact. This interaction was enhanced artistically by adding a glow effect to the tile the mouse was over, indicating to the user exactly where their cursor was on the isoX, isoY plane.

References

Figure 7. Illustration of the interaction grid. Image taken from Embers of Babylon, an interactive work built using the first prototype of our isometric world-builder engine.