The challenge with shading crowds is creating a population that is believably complex and detailed in as little time as possible. During the making of Pixar’s WALL·E, we were tasked with filling a city-sized spaceship with robotic and human characters. Art direction called for visual uniformity, showing conformism had set in amongst characters marooned in space for 400 years, but with enough visual difference to create a believably rich population. On top of the visual requirements, we needed to shade the 200+ characters quickly, ensure render efficiency, and work with a reduced set of our shading tools. To do this, we developed a flexible global shading palette, an automatic approach for driving shading features based on curvature and gprim proximity, and a process for baking customized surface details into a more efficient data set.

1 Designing A Global Palette

In WALL·E, the Axiom cruise liner is staffed with hundreds of service robots divided into utility, economy and luxury classes. Each bot’s various metal, plastic, rubber, and glass components have finish qualities representing their social class. At Pixar, each shading artist usually develops their own SLIM network, providing unrestricted creative freedom. In order to minimize complexity and enforce consistency, we created a global palette of surfaces which our artists could reuse, achieving a custom look by adjusting preset controls. For example, our metal template provides variable controls for how the metal is scuffed, rusted, chipped, bronzed, oxidized, faded, greased, dirtied, and painted. Our goal was to provide enough flexibility so that all classes of a material, plastics for example, could be created using a single shader. As shading artists developed instances of the materials we liked, we augmented the global palette with these. Eventually, shading a robot was as easy as picking from our off-the-shelf materials and tweaking a few variables.

2 Using Signals To Drive Details

Painting detail on all of our characters also proved prohibitively expensive. Most of the 200+ characters were built from over 100 unique primitives, many of which were referenced from dozens of external files. We devised a way to automatically generate two types of presence maps based on geometric features of the bots. The first, blurred curvature, was created by baking the absolute value of the surface’s second derivative into a pointcloud. Processing the pointcloud into a uniform voxel space and blurring using a fixed kernel allowed us to blur a signal which we normally can’t in our scanline renderer, Renderman. Blurred curvature (Fig. 2a) is a great signal for driving features such as scratching, rust, and chipping on metal, saturation on plastics, and thickness in glass and plastic clearcoat. The second, proximity occlusion, was generated using Renderman’s ray traced occlusion with the bot in a default pose, and then baked into a UV texture. Proximity occlusion (Fig. 2b) is a great signal for representing features like wear between moving parts, less fading in plastic, and the presence of grease and dirt. Combined with varying weights of the dot product of the surface normal against an upward facing vector, as well as chaotic fractals, these two presence maps would give us a believably rich set of details (Fig. 2c) unique to the geometry within 1 hour of beginning shading.

3 Baking Painted Features and Graphics

Even with automatic feature discovery, there were still cases where we needed to paint custom presence maps and graphics. Pixar’s established workflow uses projection paint, per vertex paint, and UV manipulation in order to define feature location and apply graphics. However, our high-end paint tools add complexity to our characters which made rendering time and memory requirements impractical. We designed a way to allow shading artists to use our familiar tools while developing the look, but then to bake the result into a simplified representation. Once a look was approved, the artist would render in a baking mode, which would write out all of the painted signals into a set of pointclouds. These pointclouds would then be parsed into a set of UV map textures. We were thus able to avoid the render-time expense of our paint tools and related shader code, but still achieve a sufficiently refined final look.

4 Lessons Learned

At first, we were concerned that these approaches to shading crowds would limit our shading artist’s creative control and the level of detail complexity. We discovered that by maintaining less code and keeping the workflow simple, consistent, and efficient this gave us enough time to address iteration, hit the production goals, and in the end populate the Axiom.