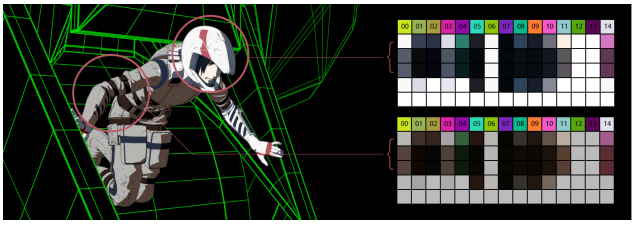
Space–Time Varying Color Palettes

Bo Zhou*
J Cube Inc.

Paolo Berto Durante†
Polygon Pictures Inc.



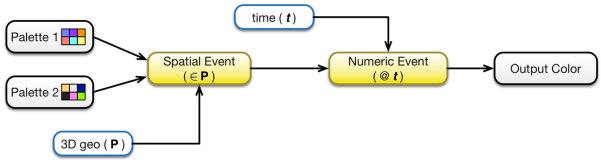


Figure 1: *Color palettes & strategy tree. Knight of Sidonia ©T. Nihei · Kodansha / KOS Prod. Committee*

1 Abstract

We improve on the widely used *color palette* technique in the context of *anime*-style rendering. We introduce an *event–driven* method to dynamically change color rendition in such rendering scenarios. Changes can happen over time or over space and are triggered by pre-defined events. Such events cause the selection of a particular *color palette* and node–based "strategy trees" help artists to plan and to visualize resulting look changes interactively. As an optimization, we rely on dynamic shading language code generation to prepare colors before rendering, avoiding the limits of render–time methods.

2 Overview

In the real world, perceived color is the result of a complex interaction between lighting conditions and material properties. In the surreal world of *anime*, illumination is local, each character has its "own" light, and surfaces are either lit or shadowed with no in–between values. Furthermore, designers explicitly choose lit and shadowed colors to define the visual perception of a material in a specific lighting situation [Brunik 2014].

Color palettes bundle the whole atlas of colors for each character and lighting condition (e.g. daylight, sunset. . .) so multiple palettes are required for a single character. Palettes can be shared across multiple shots so that film-long modifications are easily applied by updating one palette — an important productivity feature.

A representative example is one where an object enters a region of space in which a look change is required: one palette then defines the look inside the region, and another one outside of it (Figure 1, *left*). The usual (brute-force) solution would render twice, alternating the palettes, along with a mask that defines such spatial region, then blend the two in compositing. This obviously involves manual work, additional overhead and image processing.

3 Implementation

We reduced all production situations requiring colors to be selected from different color palettes under two possible cases:

1. Timed events — Such as a spaceship gate opening (our example), an explosion or a gunshot being fired at a specific time/frame. These events are triggered by animation curves and we represent them as logical operations between "current" time and event's time;

2. Spatial events — Our example of a character entering a particular scene region, in our case the outer–space of a spaceship, or a room lit by a fireplace or an animated spotlight projector: we represent regions using either pre-defined primitives (half-space, box, cone, sphere) or any arbitrary geometric object.

We use a node-graph editor to organize and prioritize these events and we call the resulting graph a "strategy tree" (Figure 1, *right*). Such strategy trees allow artists to design lighting condition using user friendly *visual programming* concepts.

The low-level part of the implementation is a custom shading language function[1] that samples palette textures at precise locations and stores resulting colors in a hash table. This function is invoked by a dynamically generated shader, which translates and evaluates the strategy tree *prior* to rendering and returns the color at render-time when the shader is executed for the actual rendering.

Dynamic code generation and pre-render palettes evaluations allow for useful optimizations. For example, timed events return values that are constant throughout a frame [2] and these constants enable the compiler to perform *constant-folding* optimizations. This usually result in very compact code and fast render times.

4 Conclusion and Future Work

We implemented this method at *Polygon Pictures Inc.*: the strategy tree allowed artists to easily and intuitively setup look-changing events and to switch between different palettes on-the-fly, using visual programming and with interactive results in the DCC application. Performance-wise, dynamic code generation allowed us to improve previous render–time methods by avoiding the need to hold palette resources in memory. Most importantly, our method betters the brute-force techniques by not requiring any additional post-processing — an important cost-saving measure. We plan to improve our system with screen-based events such as masking regions. Another area of research is automatic generation of palettes using predefined "moods". Evaluation and interpolation between such moods would allow us to generate palettes using human language semantics instead of the usual PhotoShop work.

References

BRUNIK, KAITLIN L. & CUTTING, J. E. 2014. Coloring the animated world. 128.

*bo@j-cube.jp
†paolo@ppi.co.jp

[1] A *3Delight*/RenderMan DSO *shadeop*

[2] Motion blur is an exception but not so important in our *anime*-style renders.