The Art of Demo Making

Key Ingredients

- Code
  - Provides the demo's content
- Design
  - Makes the content look good

Design

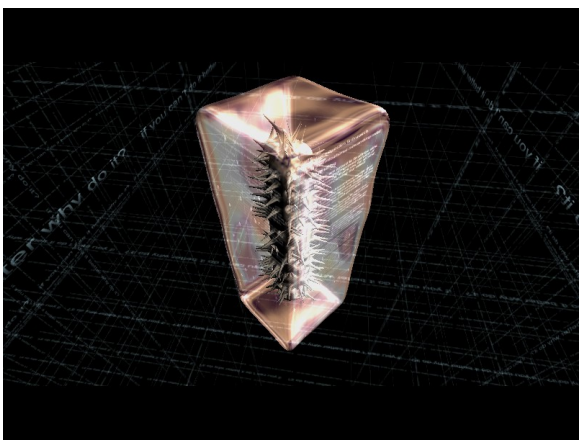
- Elements of Design
  - Basic building blocks
- Principles of Design
  - Combinations of elements
- Composition
  - Overall objective

Elements of Design

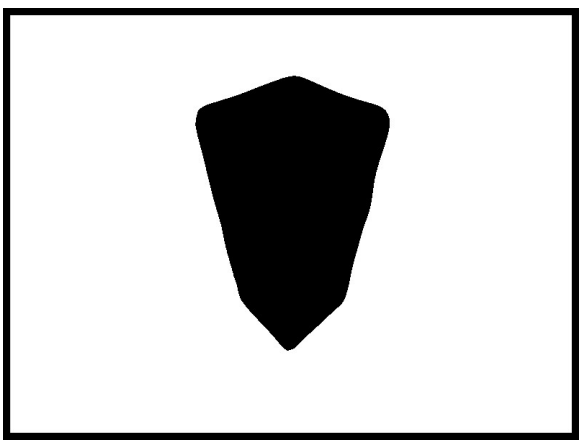
- Shape
  - Direction
  - Size
- Space
  - Color
  - Texture

Shape

- An area defined by a border
  - In design, 3D models act as 2D shapes



=

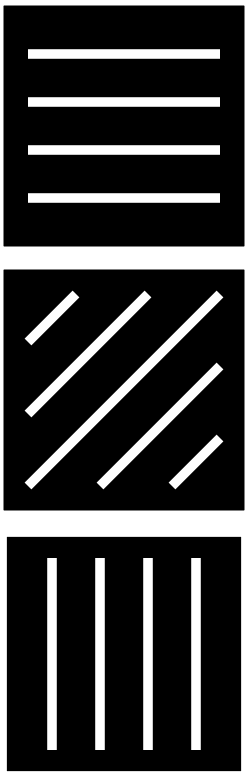


Direction

- The angle at which a shape's dominant lines flow
  - Different directions have a different “feel”

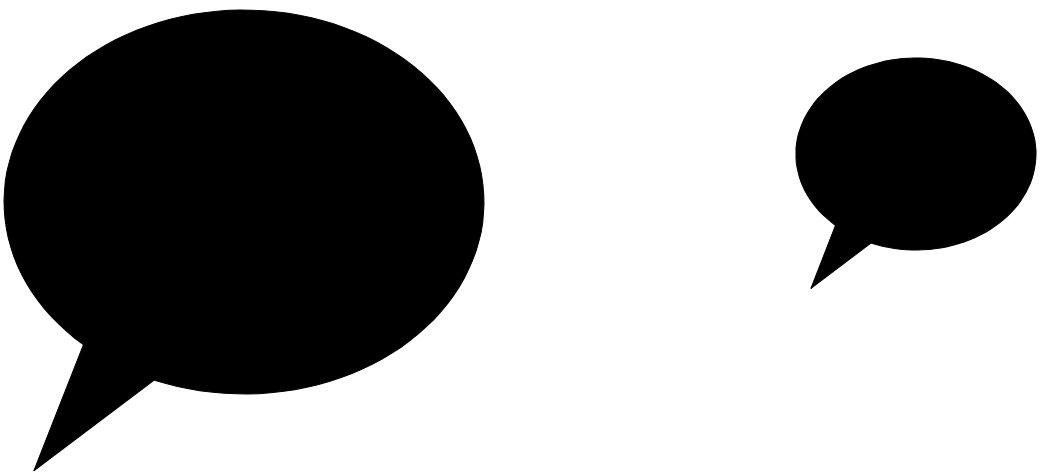
Direction Effects

- Horizontal
  - Stable, calm
- Slanted
  - Dynamic, chaotic
- Vertical
  - Alert, formal



Size

- A shape's relative scale
  - Larger objects have more visual “weight”



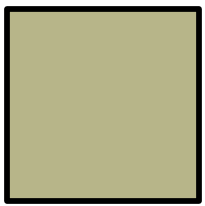
Space

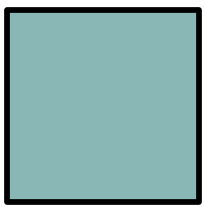
- The area either within a shape or surrounding it
  - Positive space is filled by shape
  - Negative space surrounds a shape

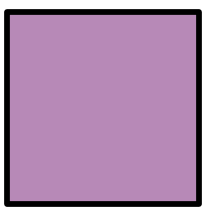
Color

- The hue, saturation, and lightness of an area
  - Humans perceive colors in HSL, not RGB
  - RGB is an irrelevant technical detail

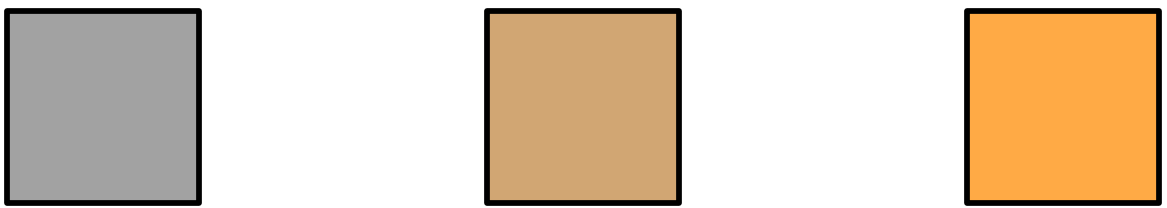
Hue



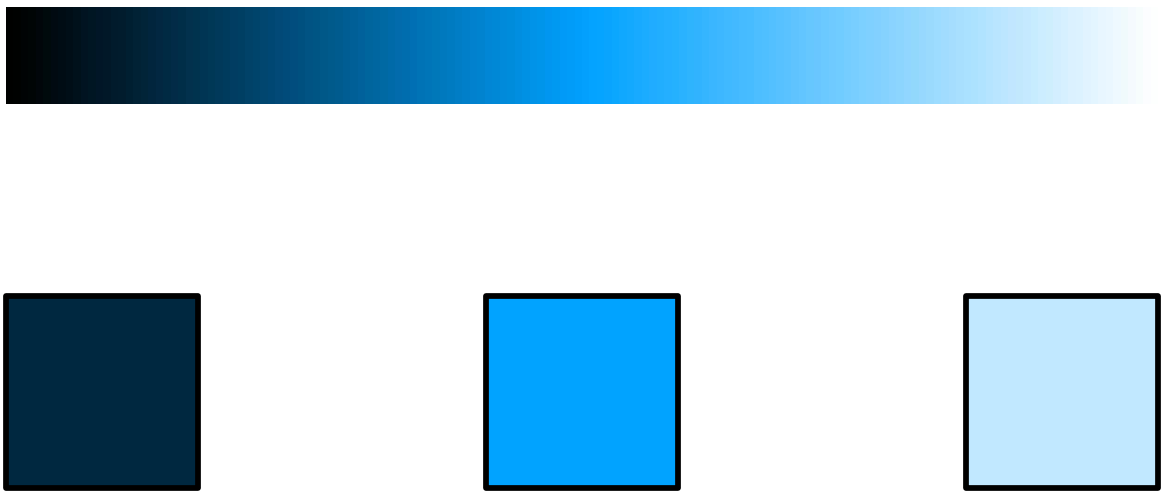




Saturation



Lightness



Lightness

- Lightness affects the visual weight of a shape
  - Dark = Heavy
  - Bright = Light

Complements

- Colors with the same saturation and lightness but 180° separation on the color wheel

Harmonics

- Colors with the same saturation and lightness but 120° separation on the color wheel.

Texture

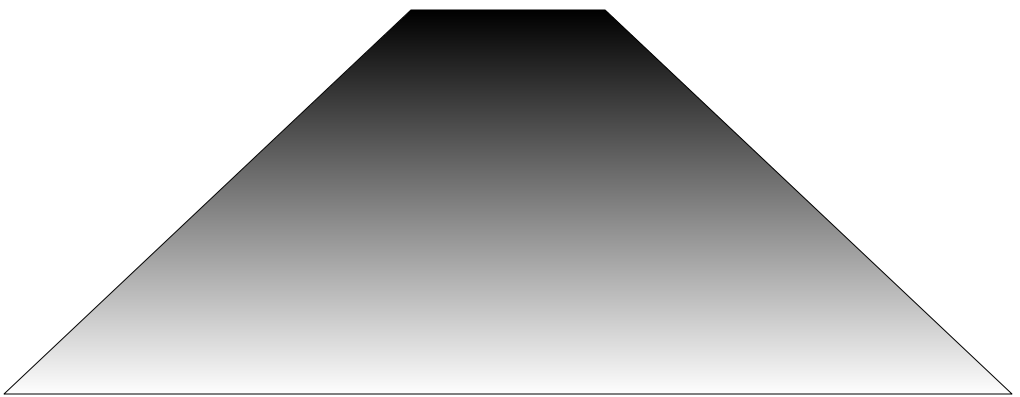
- A pattern applied to an area to create the impression of a specific tactile feel
  - Slightly different from 3D texture, which gives the impression of a specific material

Principles of Design

- Graduation
- Balance
- Contrast
- Repetition
- Dominance
- Unity

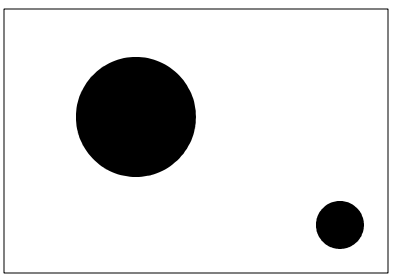
Graduation

- The smooth variation of an element through space
  - Causes the eye to move along a shape



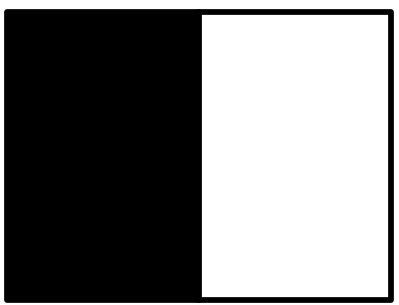
Balance

- The feeling of visual equality in a scene
  - Created with position and visual weight



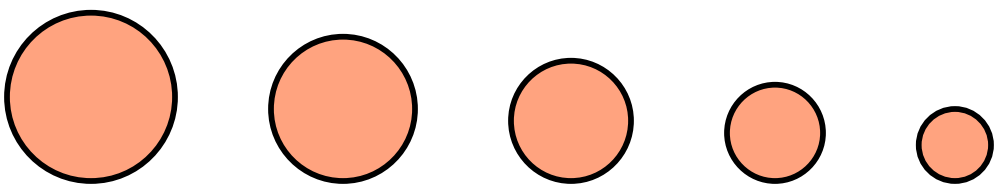
Contrast

- The juxtaposition of opposing elements



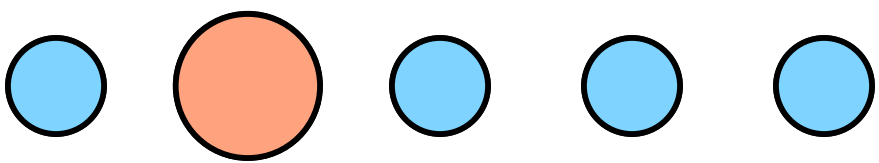
Repetition

- Multiple appearances of the same element



Dominance

- The emphasis of one object over others



Unity

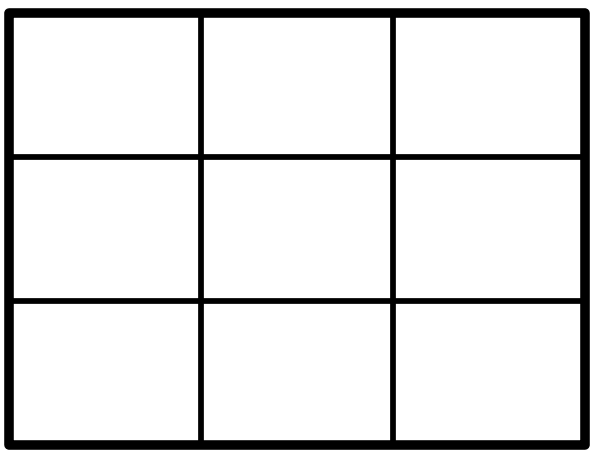
- A common connection between elements in a scene

Composition

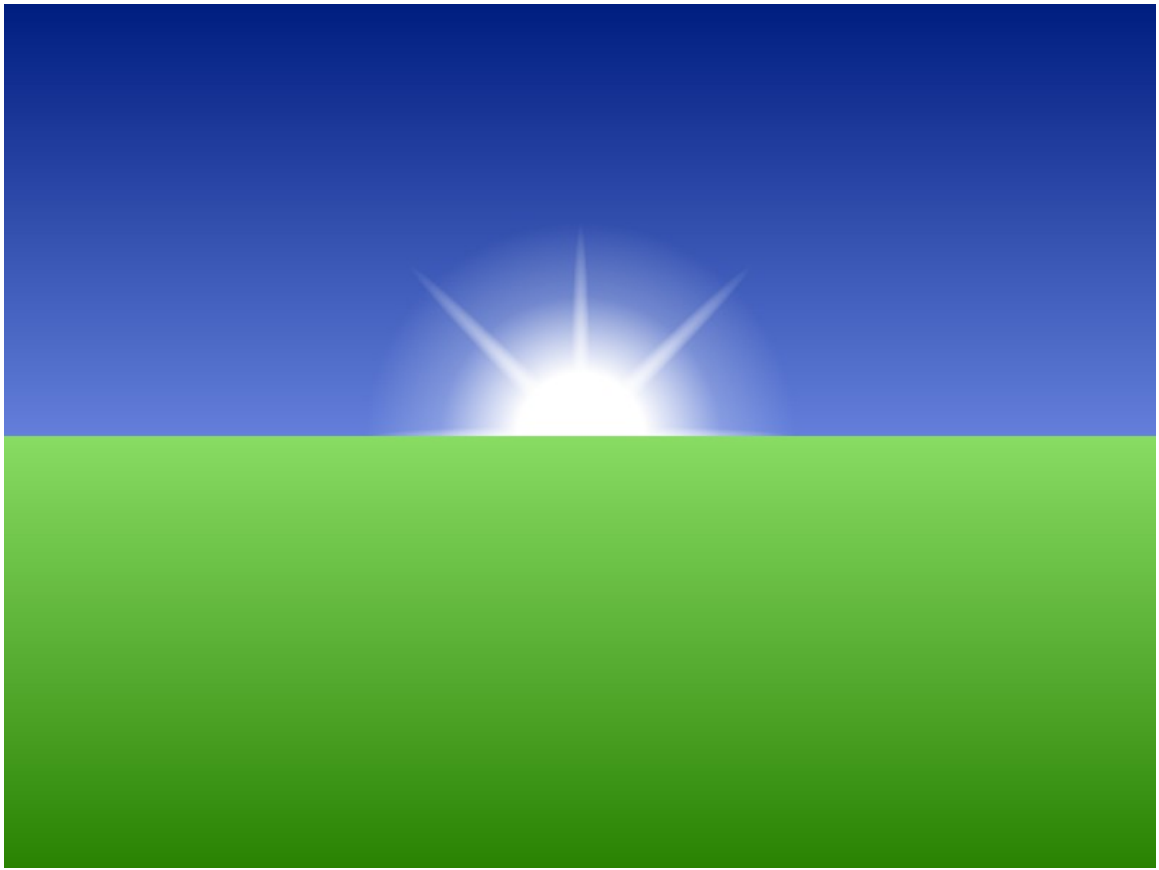
- Rule of thirds
  - Drives scene layout
- Ten-second rule
  - Guides scene timing

Rule of Thirds

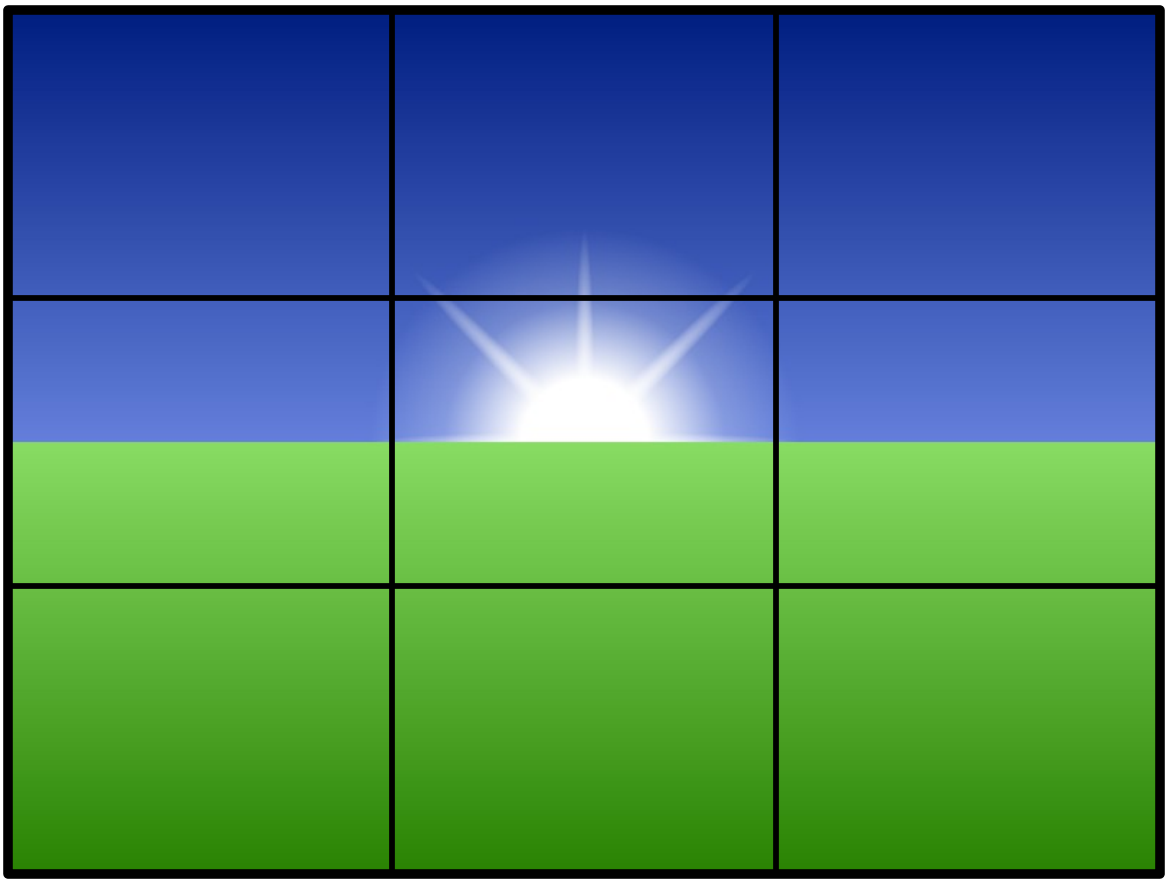
- Align elements in a scene with an imaginary three-part grid



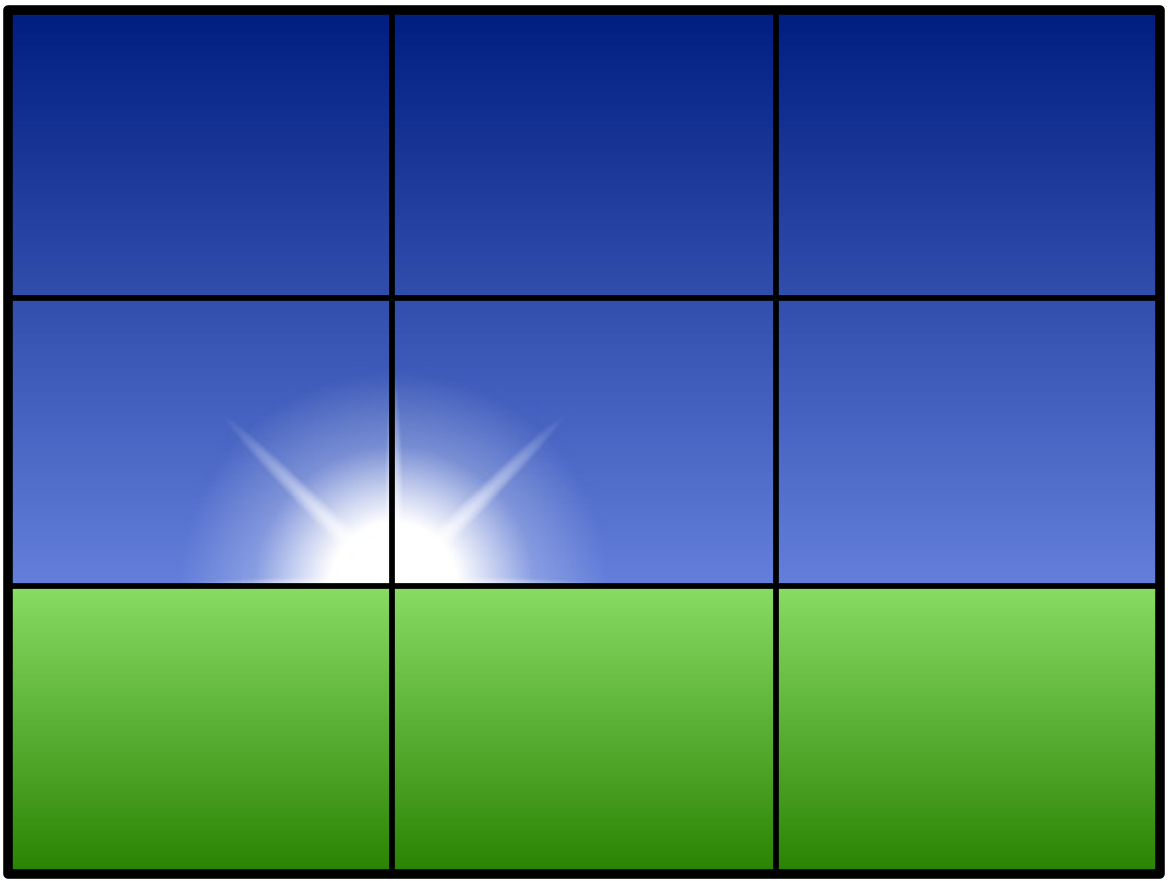
Bad Example



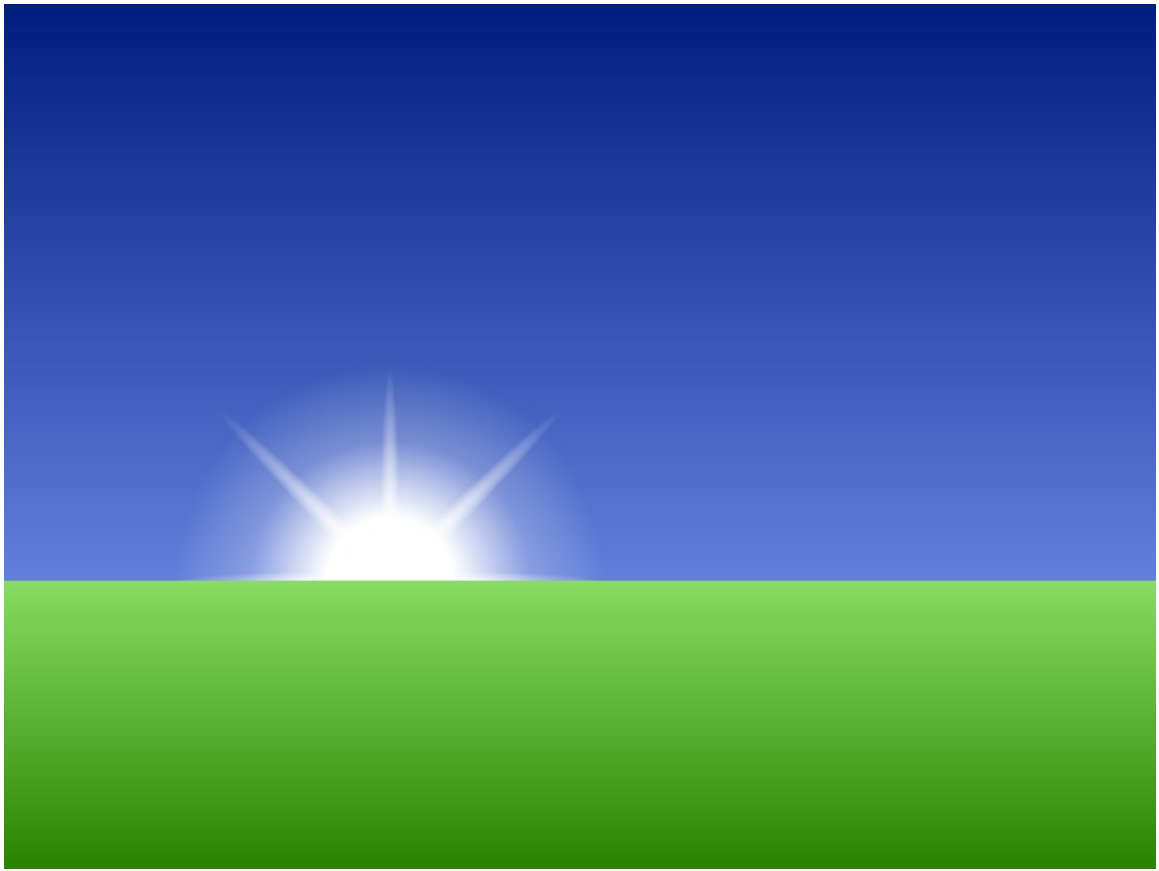
Bad Example



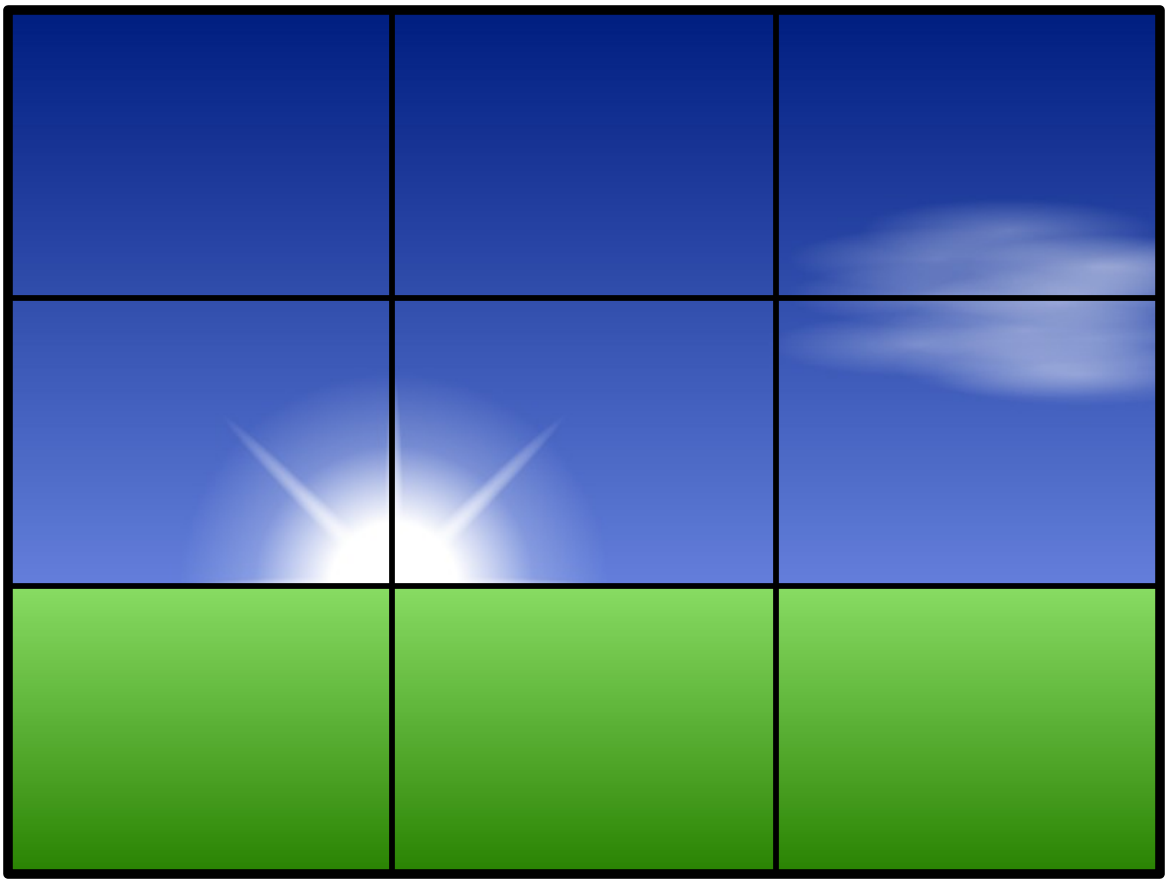
Better Example



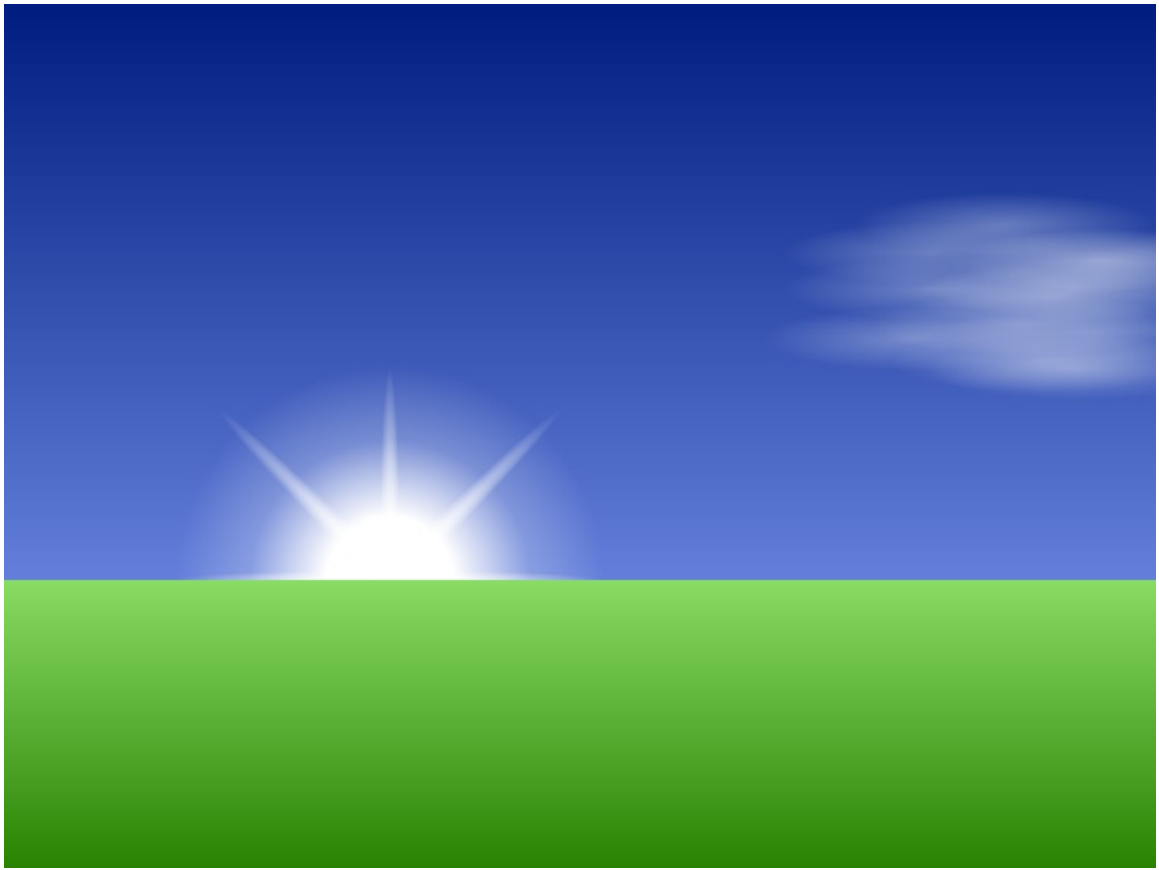
Better Example



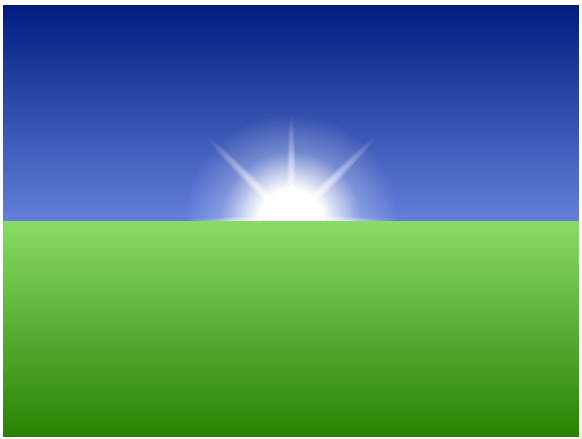
Even Better Example



Even Better Example



Comparison



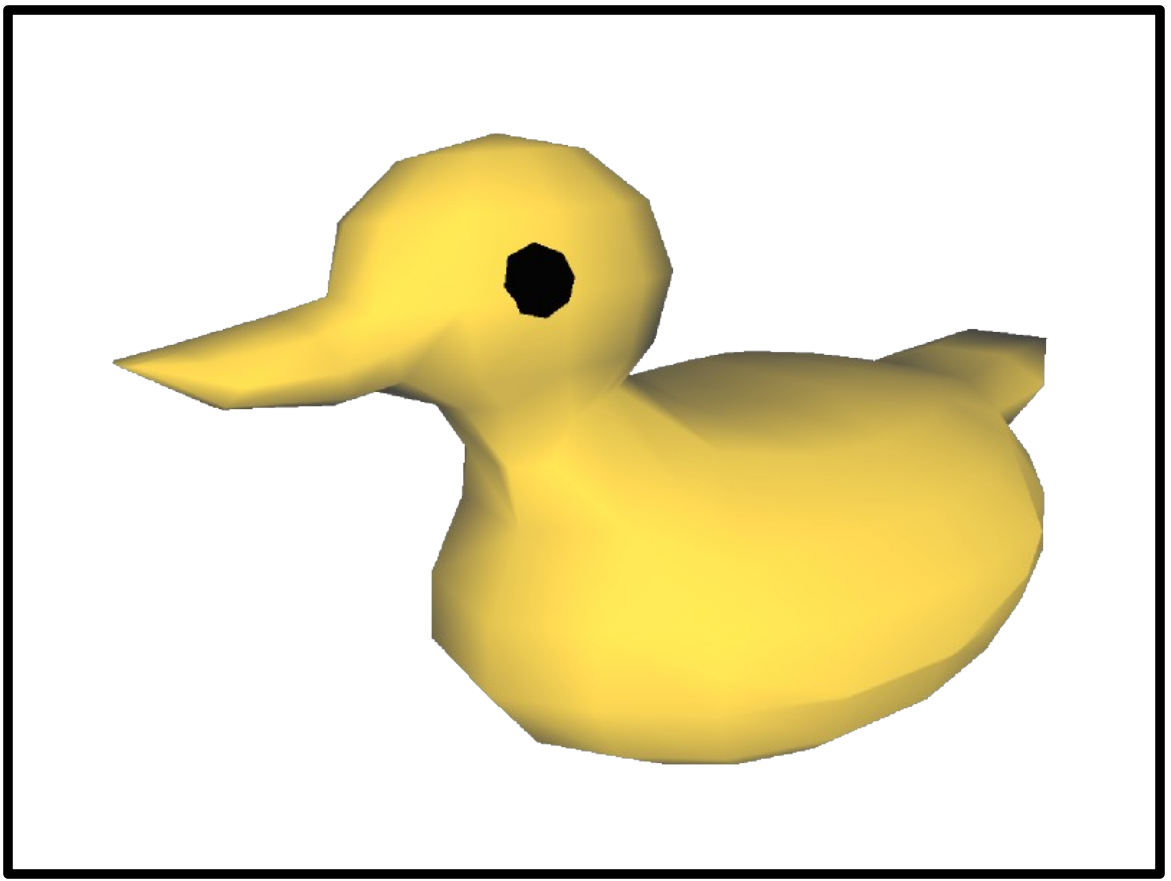
Before

After

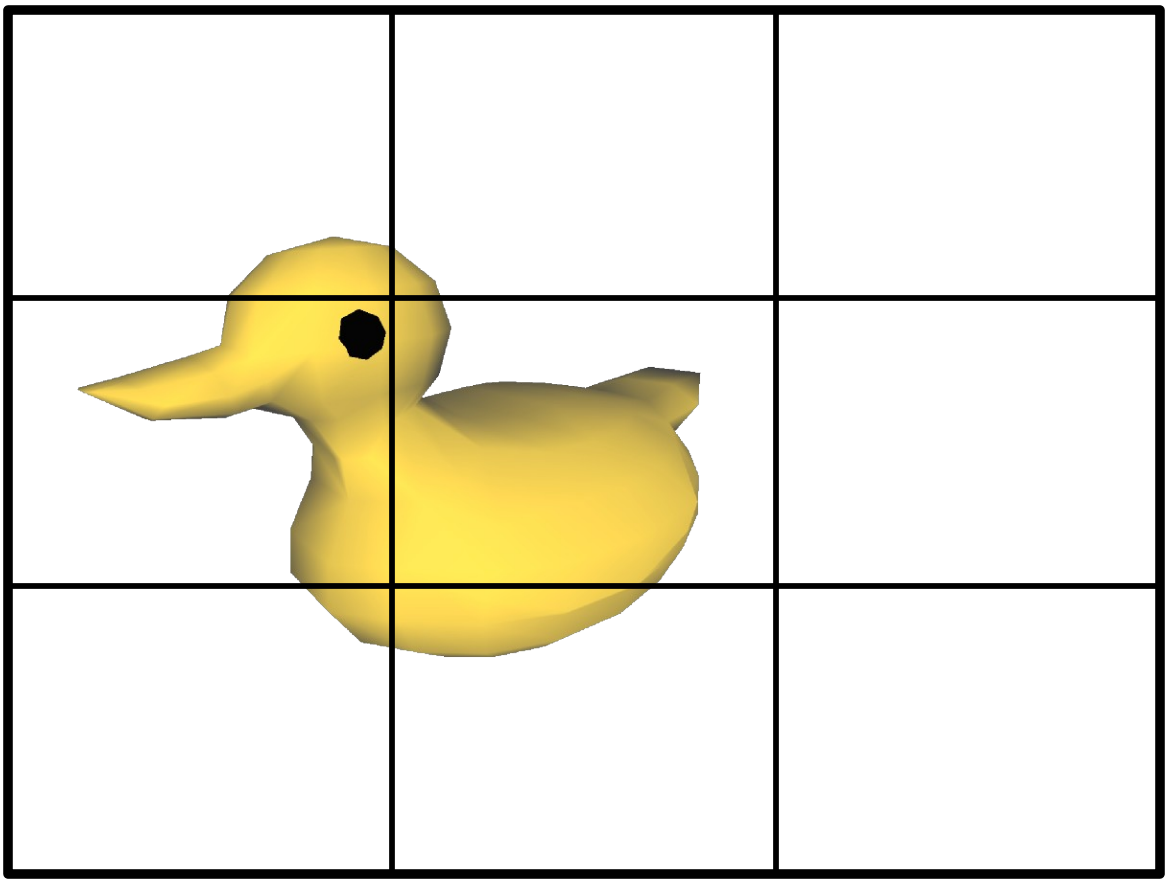
Ten-Second Rule

- Center of attention should change every 10 seconds
  - More often = hectic
  - Less often = boring
  - Effects can run for longer
  - Use visual design concepts to control center of attention

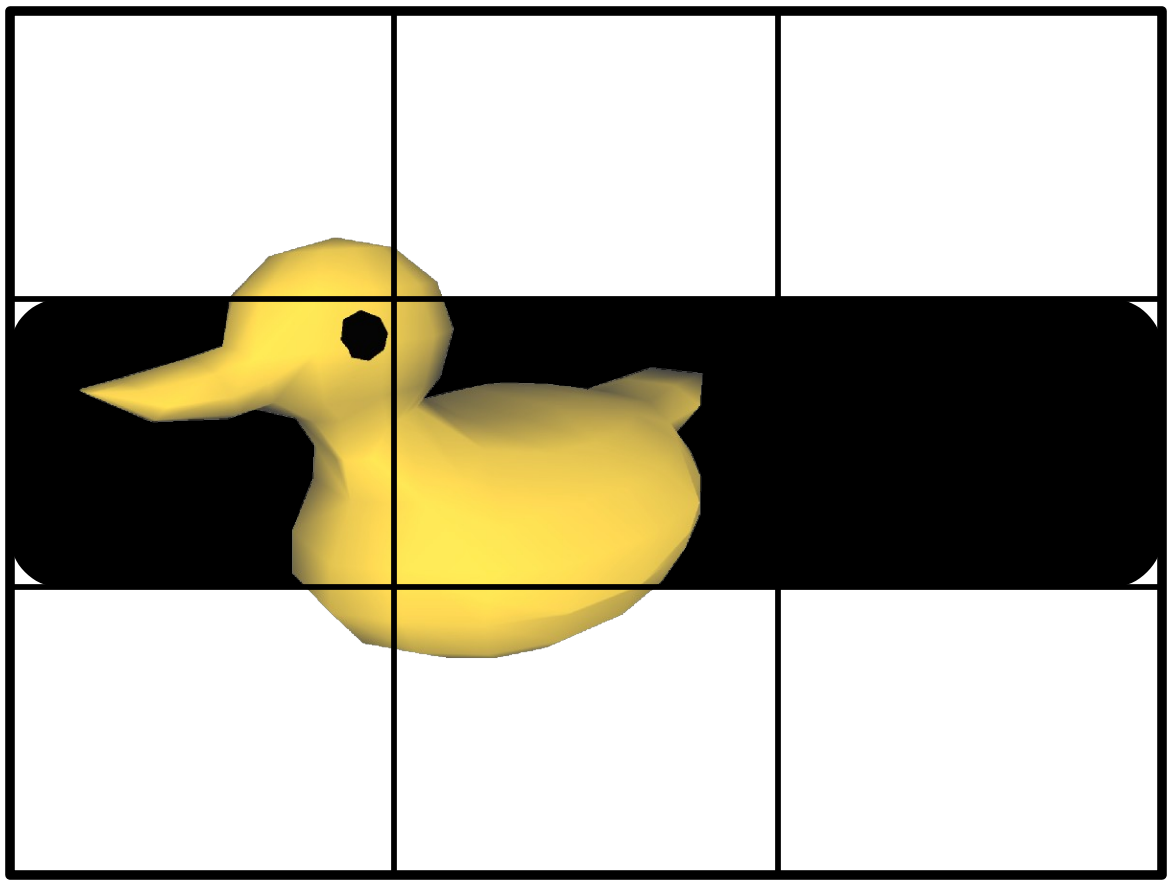
Rotating Duck Example



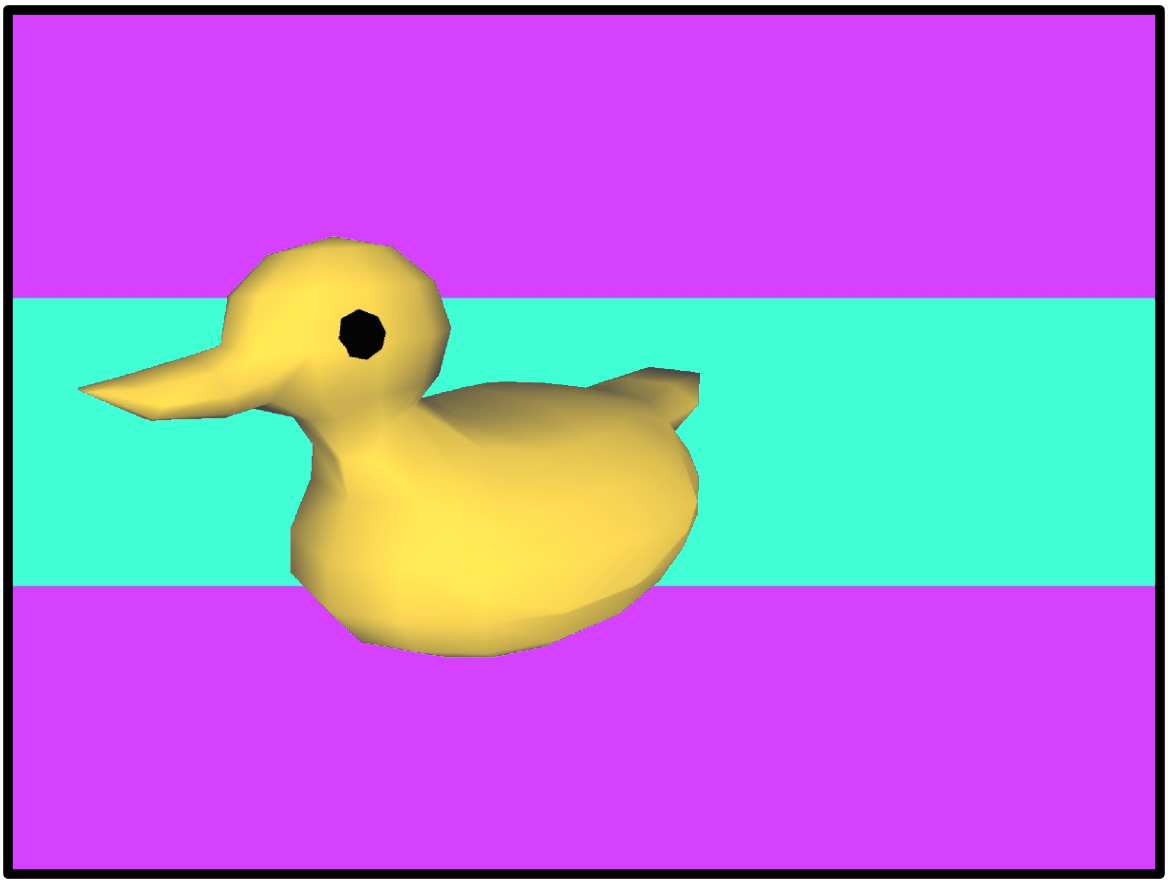
Rule of Thirds



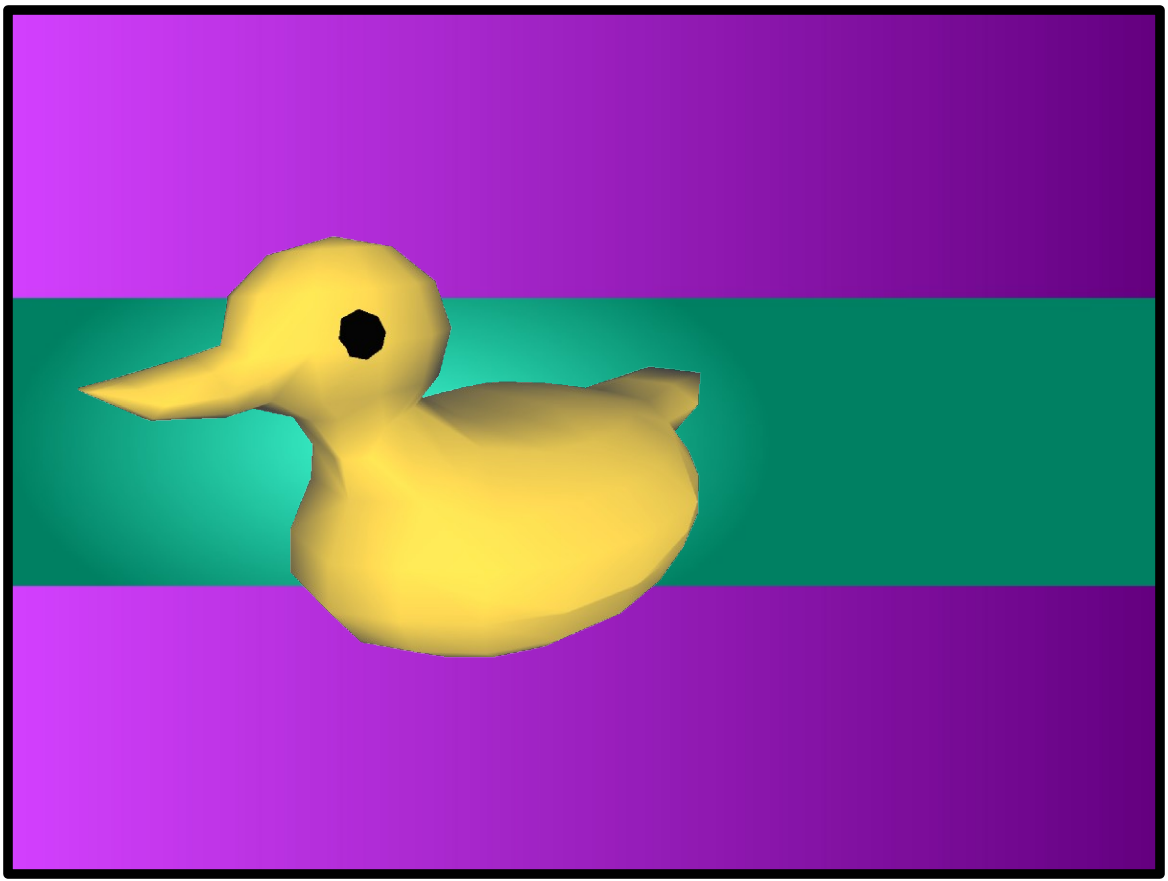
Emphasis / Shape



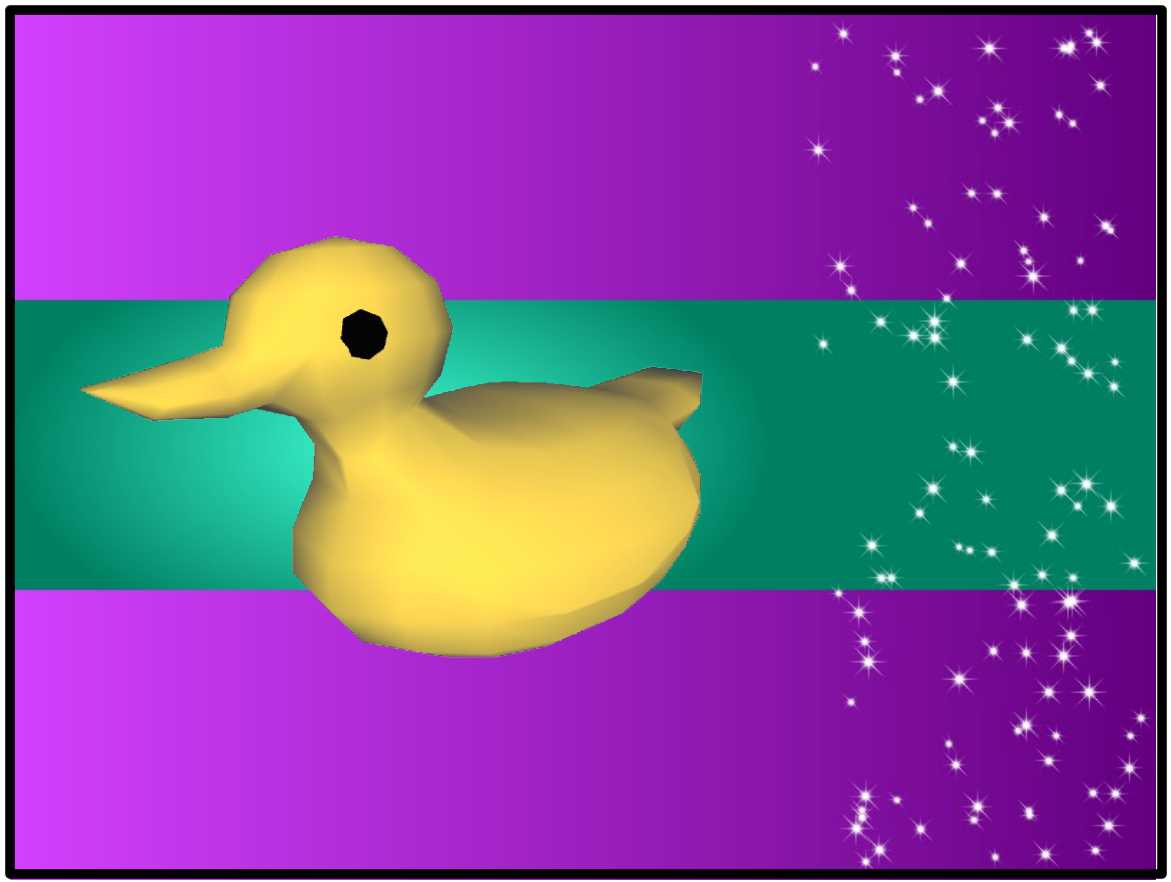
Harmonic Colors



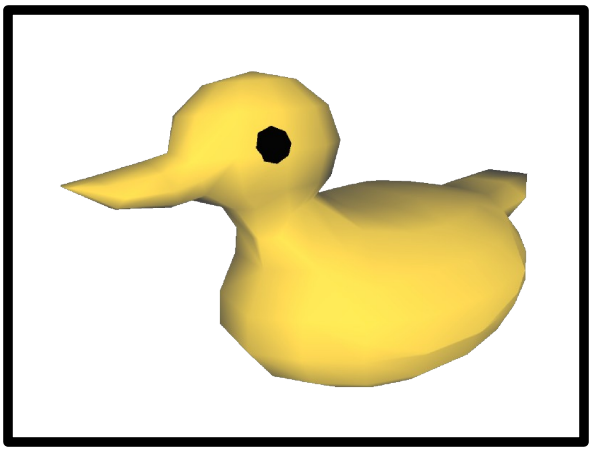
Balance / Gradient



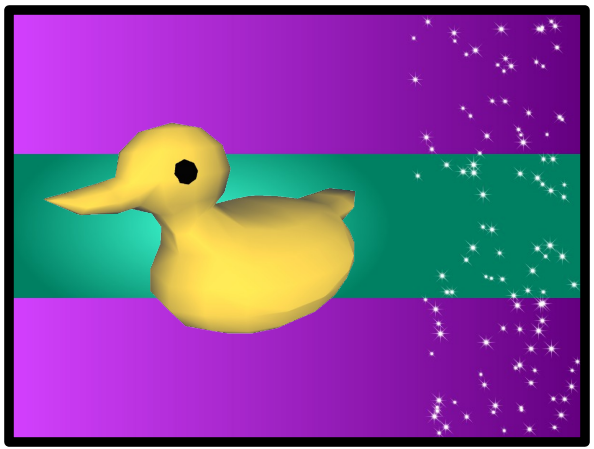
Ten Second Rule



Comparison



Before



After

Warning

- Rules are tools
  - Use them when they make sense
- Good effects are necessary
  - Design simply makes them more attractive

Code

- One objective
  - Write good-looking effects quickly and efficiently

Code

- Two key challenges
  - Arise when building complex effects
- Two general solutions
  - Determine the amount of up-front work required

Demo Objects

- The smallest units of data that have independent loading or generating code

Examples

- Textures
- 3D models
- 2D overlays
- Camera paths
- Music
- Hard-coded effects

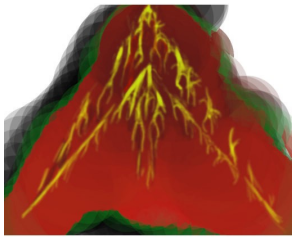
Two Key Challenges

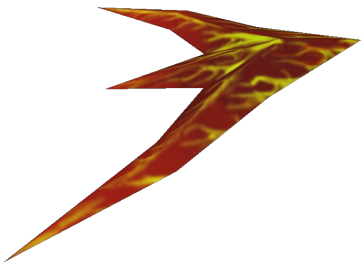
- Loading demo objects
  - Where?
  - What parameters?
- Connecting demo objects
  - Objects need each other at render-time

Example

Who loads the model?

Who loads the texture?





What joins them at render time?

Related Problems

- Memory management
- Scene composition
- Demo scripting
- Object editing
- Artist accessibility

Two Solutions

- Ad-hoc loading and creation code
- Demo object manager

Ad-Hoc Loading

- Each scene typically has its own code
- Scene code loads objects using library functions
- Object connections happen within the scene code's render function

Case Study

- Pilgrimage 2004 Invitation
  - 3D museum environment
  - Effects on walls
  - Two internal effects:
    - Butterfly
    - Cube

Program Flow

- Main loop
  - BSP render function
    - Wall effect render on an as-needed basis using visibility culling
    - Internal effect render
    - Main environment render

Advantages

- No overhead
  - Code is 100% visual-related
- Editing still possible
  - Demo uses custom camera editing system
- Flexible
  - Entire 4K intro imported as wall effect

Disadvantages

- Rigid design
  - Optimized for the particular demo, but not useful for other projects
- Lots of code
  - Each wall effect needs its own source module

Object Manager

- Centralized manager with links to all data types
- Emphasizes reusable demo objects
- Demo description file drives loading and rendering process

Case Study

- Pilgrimage 2005 Invitation
  - 3D flyby through two environments
  - Effects on walls
  - Various internal effects
    - Clouds
    - Arrow
    - Birds
    - etc...

Program Flow

- Object manager parses XML file for object ID's
- GUI subsystem requests “Main” object
- “Main” object's loading code requests additional objects recursively

Advantages

- Demo description in one file
- Allows the GUI to present a unified list of objects.
- Reduces memory management headaches
- Completely re-usable base system

Disadvantages

- Source Overhead
  - More non-visual management code
- Initial Investment
  - Eight months development time

Object Types

- ioVisual – 2D items
- ioModel – 3D items
- ioTexture – Bitmap data
- ioPath – Animation splines

Bridge Objects

- ioVisualViewport
  - Renders 3D content to a 2D area
- ioTextureTarget
  - Renders a 2D scene to a texture

Final Advice

1) Focus on effects
2) Use design principles
3) Structure your engine to do 1 and 2 as simply as possible