CS 580 – Discussion 9
HW 5

10/18/2016

Matthias Hernandez

This week

- Answers to previous questions
- HW 5 key points
- HW 5: Tutorial on Julia set: how to make a cool procedural texture
- Q&A

How does regrading of HW1-3 work?

- We will apply the grading rubric of HW1-3 to the code submitted for HW4 if you requested regrading.

- The updated grade will been added in a new column on blackboard. We will keep both grades.

- If you haven't requested re-grading and want it, please send us an email before Friday 10/21.

Does everyone have a group for the final project?

- Project proposal is due soon.

- If you have no group or a group of 2, please email the TAs.

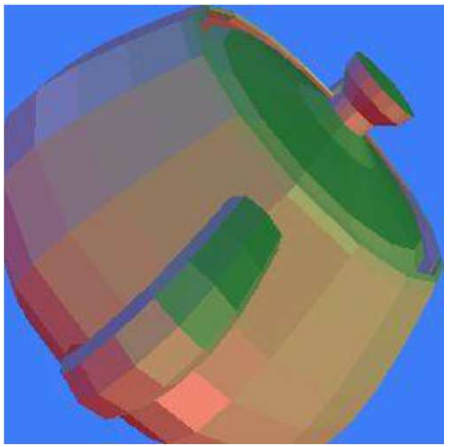
HW4 questions

- What is the default camera? Camera parameters to be used if no camera defined in the app.
  - Location, look-at point, world-up defined when initializing the display

- How to use it?
  - Do not define any camera parameter in Application4.cpp.

HW4 questions

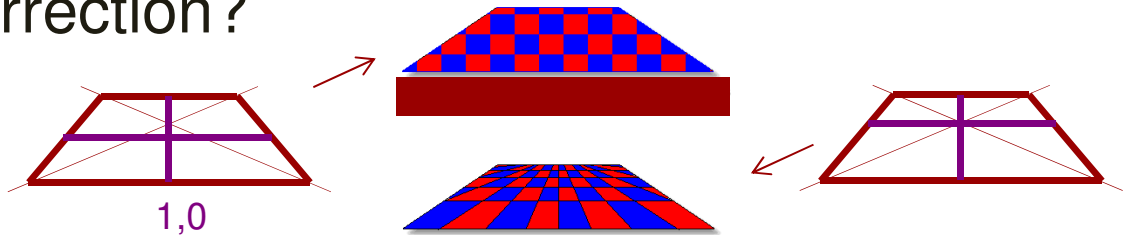
- How to apply flat shading?
  - 1. Apply a pre-defined constant color to each face.
  - 2. Compute a normal to each triangle and use it with the shading equations.



  - Add line “interpStyle = GZ_FLAT” in Application4.cpp

HW5 questions

- Why do we need to worry about perspective correction?

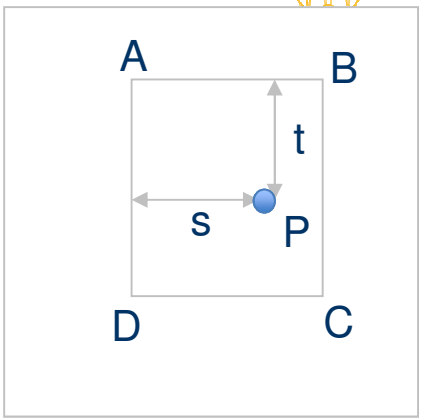


Linear Interpolation

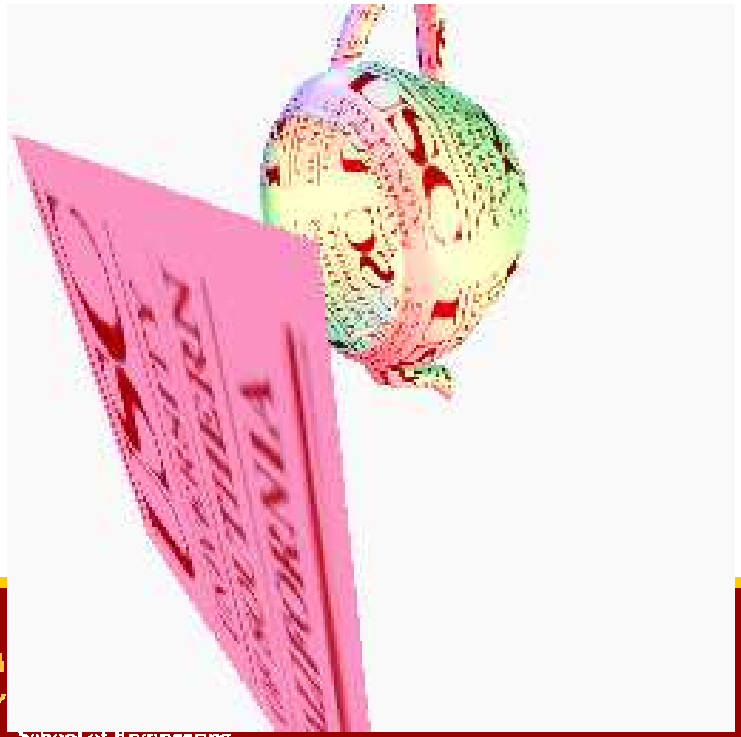
Perspective Interpolation

HW5 questions

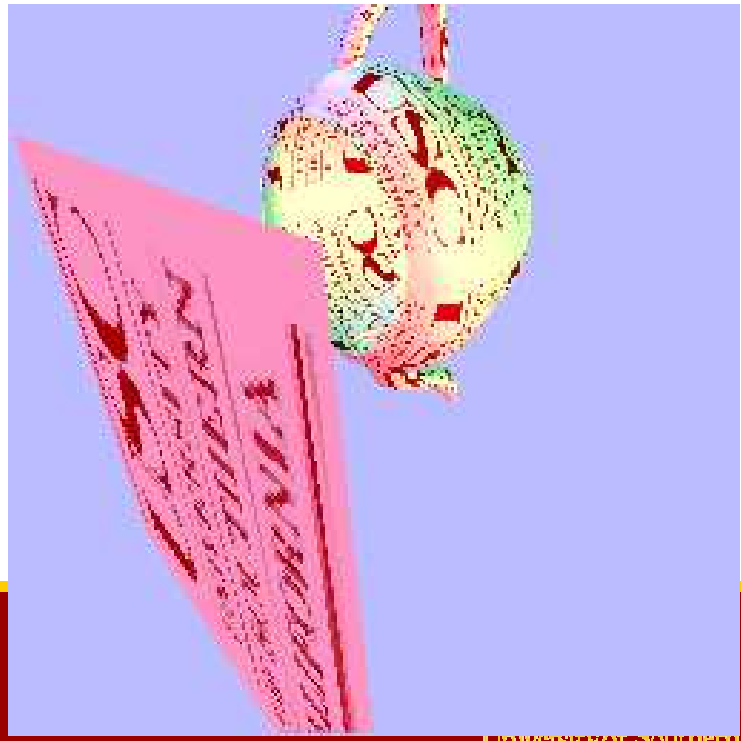
- Why do we need to worry about bilinear interpolation in the image texture?



With bilinear interpolation



Without interpolation



HW5 questions

- Where do we use the texture value K_T?

- Phong shading: replace $K_d = K_a = K_T$

  $C = (K_S \Sigma_L [Ie\ (R \bullet E)^s]\ ) + (K_T \Sigma_L [Ie\ (N \bullet L)]) + (K_T\ Ia)$

material attribute (fixed) texture lookup (pixel-by-pixel)

- Gouraud shading: replace $K_T = K_S = K_d = K_a$

  $C = (K_T)\ (\Sigma_L [Ie\ (R \bullet E)^s] + \Sigma_L [Ie\ (N \bullet L)] + Ia)$

texture lookup (pixel-by-pixel) compute at vertices and interpolate to pixels

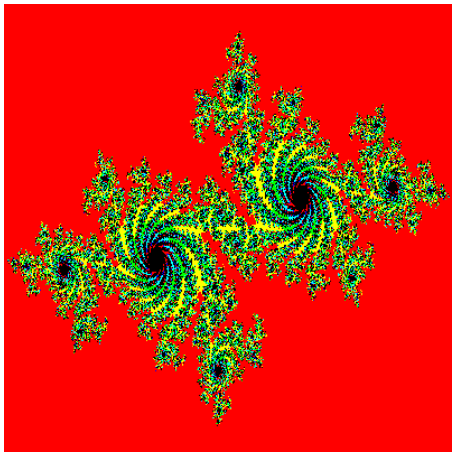
HW5: Julia set tutorial

- How to make a Julia procedural texture?
  - $X = u + i\,v$
  - Define $F(X)$ ← Your procedural value
  - Define $G: [0; 1] \rightarrow [0; 255]^3$ ← Your color mapping

Example:

$F(X) = X^2 + (-0.7 + i\,0.27015)$

$G(X) = \text{hsv2rgb}((X \,\%\, 256)/255, 1, 1)$



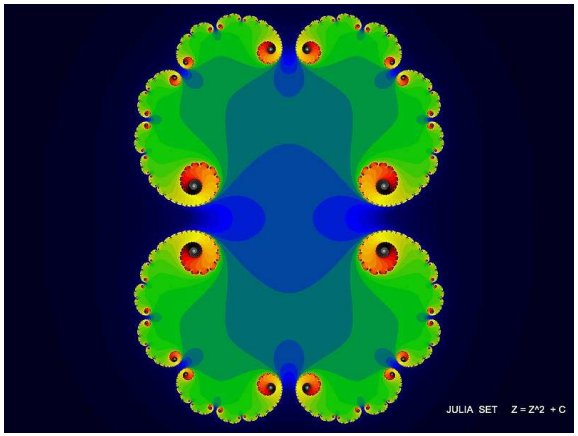
HW5: Julia set tutorial

For each pixel (u,v) of your generated texture image:

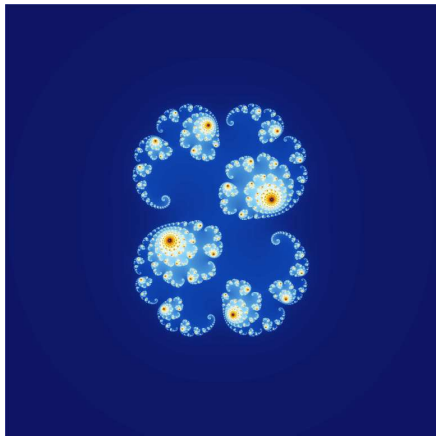
- Set X = [(u-W/2) / (W/2);
  (v-H/2) / (H/2)]
  ← [W H] is the size of your texture
- Apply N times:
  ← N is your maximum number of iterations
  - X=F(X)
  - If length(X) > 2
    - break;
    ← You get an index i after which the length of X is larger than 2
- Compute Z = i / N
- Get the color G(Z) = [R(Z), G(Z), B(Z)]
- Put that color at pixel (u,v)

HW5: Julia

$F(X) = X^2 - 0.8+0.156i$



$F(X) = X^2 + 0.279$



$F(X) = X^2 + 0.285+0.01i$

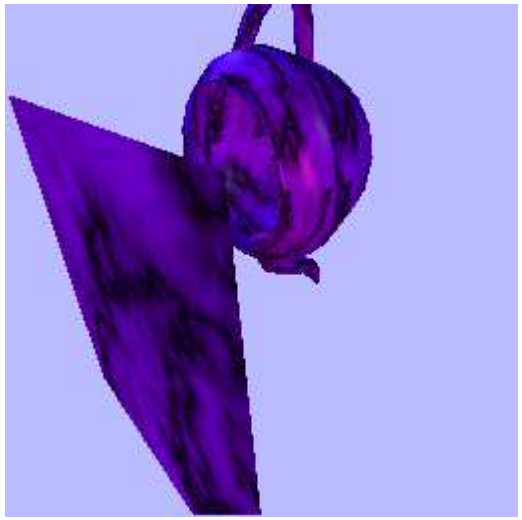
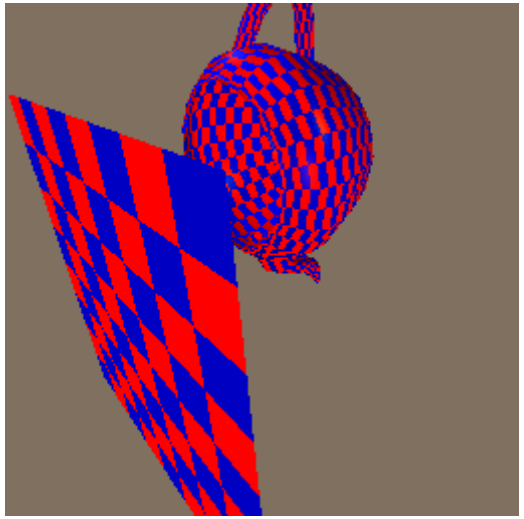
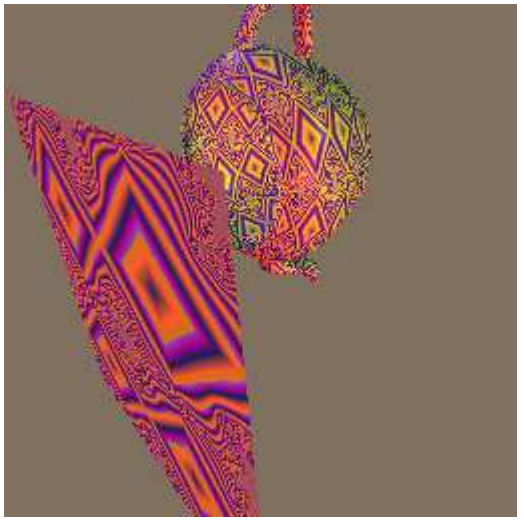
$F(X) = \exp(X^3) - 0.621$

$F(z) = X^2 * \exp(X) + 0.21$

Images from Wikipedia.org/wiki/Julia_set

HW5: procedural

- There are lots of other procedural textures

- Be creative!

HW5: procedural texture

- How to visualize your procedural texture?
  - See your texture by replacing in Application5.cpp

  valueListShader[5] = (GzPointer)(tex_fun);

  by

  valueListShader[5] = (GzPointer)(ptex_fun);

HW5 pitfalls

- Perspective-Z: make sure the texture is not distorted on the plane and the teapot.
- Bilinear interpolation: make sure the texture is not too aliased.
- Procedural texture: do not forget to implement a procedural texture.

Q&A

HW4 questions

- How does Gram-Schmidt Orthonormalization work?

  - Given a vector space of dimension n and a set of k vectors $(v1,\ldots,vk)_{k<n}$, build an orthonormal set (e1,…,ek) that spans the same space as (v1,…,vk)

  - $e1 = v1 / ||v1||$
  - $u2 = v2 - (v2|e1)\ e1 \rightarrow e2 = u2 / ||u2||$
  - $u3 = v3 - (v3|e1)\ e1 - (v3|e2)\ e2 \rightarrow e3 = u3 / ||u3||$
  - …

HW4 questions

- Given vector v1 = [2; 2; 1]
  v2 = [3; 0; 3]
  Build (e1,e2) such that e1 $\perp$ e2 and ||e1||=||e2|| = 1
  - e1 = v1 / ||v1|| = [2/3; 2/3; 1/3]
  - u2 = v2 – (v2|e1) e1
    - (v2|e1) = 3*2/3 + 0*2/3 + 3*1/3 = 3
    - (v2|e1) e1 = [2; 2; 1]
  - u2 = [1; -2; 2] → e2 = u2 / ||u2|| = [1/3; -2/3; 2/3]

  - (e1|e2) = (2/3)*(1/3) + (2/3)*(-2/3) + (1/3)*(2/3) = 0

HW4 questions

- e1 = v1 / ||v1||
- u2 = v2 – (v2|e1) e1 → e2 = u2 / ||u2||

- Why does it work?
  - It is clear that ||e1||=||e2||=1
  - (e1|e2) $\propto$ (e1|v2 – (v2|e1)e1)
    $\propto$ (e1|v2) – (v2|e1)||e1||²
    $\propto$ 0
    → Can be proved by induction for k>2.