Rendering Mirage

Team 3 Seo Hansol, Lim Mingi
CS482 Fall 2018 Final Presentation
DEMO
Contents

● Artistic Editing For Mirage Image
● Our Idea
● Challenges
● Implementation
Artistic Editing
For Mirage Image
Refractive Index Distribution Model

Adapt **APM\[GSM*06\]** as a spatial encoding

Images from [GSM*06]

This slide is copied from slides of [Choi 17]
Refractive Index Distribution Model

New formulation suggested for the refractive index distribution: **Logistic Approximation**

\[
\begin{align*}
n(h) &= \frac{k_s}{|n_{bg}| + |n_{inv}| + |n_{hotspot}|} \\
&\quad \cdot \left( f_{\text{logistic}}(n_{bg}, a_{bg}, 0, h) \\
&\quad + f_{\text{logistic}}(n_{inv}, -a_{inv}, h_{\text{ciso}}, h) \\
&\quad + f_{\text{logistic}}(n_{hotspot}, a_{hotspot}, 0, h) \right) \\
&\quad + 1,
\end{align*}
\]

\[
f_{\text{logistic}}(L, k, x_0, x) := \frac{L}{1 + e^{(x_0-x) / k}}.
\]

\[
k = \begin{bmatrix} n_{bg} & n_{inv} & n_{hotspot} & h_{\text{ciso}} & a_{bg} & a_{inv} & a_{hotspot} & k_s \end{bmatrix}^T
\]
Spatial Encoding and Optimization

- **Cost Function**
  - For i-th pair, spatial encoding $L$, and parameter vector $k$
    \[
    \text{cost}(k) := \sum_{i} \text{dist}_{L,i}(k)
    \]
  - solve by **Regression**
    \[
    \arg\min_{k} \text{cost}(k)
    \]
Rendering method

- Ray-marching algorithm

\[ \frac{dv}{ds} = \nabla_x n, \]
\[ \frac{dx}{ds} = \frac{v}{n}. \]

[ABW14]
Limitations

- This system is focused on creating static scenes
  - This formulation does not consider the spatial location of the hot spot
- Unintuitive UI
- No illumination model

⇒ Unrealistic mirage scenes
Our Ideas
Areal Hot Spot
Areal Hot Spot

- Hot spot is no longer depends on only height
- Hot spot has a circular shape that can be defined on the surface.
- User can modify its x, z position and radius.
- Optimize additional parameters using the existing optimizer
- By changing the formula to take into account the area of the hot spot, you can observe a light path that changes spatially.
Areal Hot Spot
Areal Hot Spot

- Save hotspot information additionally
- Sigmoid as the reduction function in distance
- We introduce new logistic formula to compute areal hot spot exactly.
- We newly compute a derivative of our new logistic function because we use RK4 method to estimate the light path.
**Areal Hot Spot**

- **important point**
  - formula 1: new hotspot
  - \((1 - \text{sigmoid}(\text{dist})) \times f_{\text{logistic}}(n_{\text{hotspot}}, a_{\text{hotspot}}, 0, h)\)
  - formula 2: derivative of new hotspot
  - previous: 1D formula\((h)\), current: 3D formula\((x, y, z)\)
  - And their counterparts in fragment shader

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n_{\text{bg/inv/hotspot}})</td>
<td>Refractive index difference caused by background/inversion layer/hot spot</td>
</tr>
<tr>
<td>(h_{\text{ciso}})</td>
<td>Height of inversion layer</td>
</tr>
<tr>
<td>(a_{\text{bg/inv/hotspot}})</td>
<td>Rate of change of refractive index for background/inversion layer/hot spot</td>
</tr>
<tr>
<td>(k_s)</td>
<td>Scale factor</td>
</tr>
</tbody>
</table>
Areal Hot Spot

\[
\frac{dv}{ds} = \nabla_x n, \\
\frac{dx}{ds} = \frac{v}{n}.
\]
Areal Hot Spot
Areal Hot Spot

```cpp
glm::dvec3 TestModel::gradient(glm::dvec3 p)
{
    double altitude = p.y;
    double deStdDeriv =
        backgroundDeriv(altitude) +
        inversionLayerDeriv(altitude) /*
        hotspotDeriv(altitude)*/;

    glm::vec3 hotspotDerivSum = glm::vec3(0, 0, 0);
    for (int i = 0; i < hotspotTotalNum; i++)
    {
        hotspotDerivSum += hotspotDeriv(altitude, p, hotspotInfo[i], hotspotNs[i], hotspotDropoffs[i]);
    }

    double hotspotNSum = 0;
    for (int i = 0; i < hotspotTotalNum; i++)
    {
        hotspotNSum += abs(hotspotNs[i]);
    }

    glm::vec3 deStdDerivNormalized = glm::vec3(0, deStdDeriv, 0) + hotspotDerivSum;
    deStdDerivNormalized /= abs(backgroundMaxN) + abs(inversionDeltaN) + hotspotNSum;

    return deStdDerivNormalized;
}
```
Areal Hot Spot

```cpp
inline double TestModel::hotspot(const double &altitude, const glm::dvec3 p, const std::pair<glm::vec3 *, float>&)
{
    //std::cout << "hotspotDist : " << hotspotDist(hotspot, p) << std::endl;
    return (1 - sigmoid(10 * hotspotDist(hotspot, p))) * logistic(hotspotN, 0.3*hotspotDropoff, 0, altitude);
}
```

```cpp
inline glm::dvec3 TestModel::hotspotDeriv(const double &altitude, const glm::dvec3 p,
{
// d hotspot / d = - 10 * L * D' * S'.10*D - S.10*D * L'
// for x and z, l'(y) = 0
    double logis = logistic(hotspotN, 0.3*hotspotDropoff, 0, altitude);
    double logisDeriv = logisticDeriv(hotspotN, 0.3*hotspotDropoff, 0, altitude);
    double dist = hotspotDist(hotspot, p);
    glm::dvec3 distDeriv = hotspotDistDeriv(hotspot, p);
    return glm::dvec3(
        -10 * logis * distDeriv.x * sigmoidDeriv(10 * dist),
        0,
        -10 * logis * distDeriv.z * sigmoidDeriv(10 * dist)
    );
}
```
Areal Hot Spot

```cpp
inline double TestModel::hotspotDist(std::pair<glm::vec3 *, float> hotspot, const glm::dvec3 p) const
{
    glm::vec3 hotspotPos = *hotspot.first;
    float hotspotRadius = hotspot.second;

    if (glm::isnan(sqrt((hotspotPos.x - p.x)*(hotspotPos.x - p.x) + (hotspotPos.z - p.z)*(hotspotPos.z - p.z))))
        return -hotspotRadius;

    return sqrt((hotspotPos.x - p.x)*(hotspotPos.x - p.x) + (hotspotPos.z - p.z)*(hotspotPos.z - p.z)) - hotspotRadius;
}
```

```cpp
inline double TestModel::sigmoid(const double x) const
{
    if (x < 100)
        return (exp(x) / (exp(x) + 1));
    else
        return 1;
}
```

```cpp
inline double TestModel::sigmoidDeriv(const double x) const
{
    if (x < 50)
        return (exp(x) / (exp(x) + 1) / (exp(x) + 1));
    else
        return 0;
}
```
UI Improvements
Better UI

- Point positioning
  - Points can be off surface
Better UI

- Free movement during point assignment
Better UI

- Camera positioning
  - "C" key for camera positioning
Challenges
Code

- Building the code was an unexpected difficulty
- Fortunately the code itself was modular enough

- Required knowledge about SDL2, OpenGL, GLSL
UI

- Original UI was not suitable for our testing
- Freer camera
- Point movement
- New UI components for hot spot handling
Numerical Error

- C++’s floating point arithmetic is not very reliable.
- \( \exp(x) / \exp(x) + 1 \) is \( \text{NAN} \) when \( x > \sim 80 \).
- Hard to find problem, better check early.

```cpp
inline double TestModel::sigmoid(const double x) const
{
    if (x < 100)
        return (exp(x) / (exp(x) + 1));
    else
        return 1;
}

inline double TestModel::sigmoidDeriv(const double x) const
{
    if (x < 50)
        return (exp(x) / (exp(x) + 1) / (exp(x) + 1));
    else
        return 0;
}
```
3-dimensional Extension

- Refractive radiative transfer equation & Runge-Kutta method from original code is dependent only on altitude (y axis).
- Our areal hot spot involves all three dimension.
- Need to understand RRTE & RK4, then extend it to 3D.
OpenGL & Shader

- Shader does not have “array-like” object
- Should use texture for variable length data (hot spots)
- 6x1024 2D texture can hold up to 1024 hot spots

```cpp
for (int i = 0; i < (model->hotspotInfo).size(); i++) {
    // position
    glTexSubImage2D(GL_TEXTURE_2D, 0, 0, i, 1, 1, GL_RED, GL_FLOAT, &((model->hotspotInfo[i].first)->x));
    glTexSubImage2D(GL_TEXTURE_2D, 0, 1, i, 1, 1, GL_RED, GL_FLOAT, &((model->hotspotInfo[i].first)->y));
    glTexSubImage2D(GL_TEXTURE_2D, 0, 2, i, 1, 1, GL_RED, GL_FLOAT, &((model->hotspotInfo[i].first)->z));
    // hotspot Radius
    glTexSubImage2D(GL_TEXTURE_2D, 0, 3, i, 1, 1, GL_RED, GL_FLOAT, &model->hotspotInfo[i].second));
    // hotspot N
    glTexSubImage2D(GL_TEXTURE_2D, 0, 4, i, 1, 1, GL_RED, GL_FLOAT, &model->hotspotNs[i]));
    // hotspot Dropoff
    glTexSubImage2D(GL_TEXTURE_2D, 0, 5, i, 1, 1, GL_RED, GL_FLOAT, &model->hotspotDropoffs[i]);
    // hotspot Temperature
}
```
OpenGL & Shader

- Shader does not have "array-like" object
- Should use texture for variable length data (hot spots)
- 6x1024 2D texture can hold up to 1024 hot spots

- Debugging shader
  - NAN, again
  - Shader cannot "print" or "log", as it is run on GPU
  - Always watch out the typo
Remaining Problems
Restrictions on Hot Spot

- Hot spot shape is currently circular only
  - Need several circular hot spots for other shapes
- Hot spot position is currently on surface only
  - Possibly hot spot can be on mid-air for more effect
  - Or optimizer may be able to optimize hot spot position
Quality of Mirage

- For high quality image, user should specify “good” pairs of source/destination points
- Optimization dilemma
  - More pairs, higher quality, slower speed
  - Less pairs, faster speed, lower quality
- Optimizer is somewhat unpredictable
  - Finding out optimizer-friendly pairs is difficult
  - Wave-like overfitting: background + inversion layer
Members & Roles

- Seo Hansol: Presentation, UI Improvement, Areal Hot Spot Improvement, Shader Implementation

- Lim Mingi: Presentation, Areal Hot Spot Implementation
Q & A
Runge-Kutta Method

t_new = t + h

Use k1, k2, k3, k4

k1 = h * y' (t, y)
k2 = h * y' (t+h/2, y+k1/2)
k3 = h * y' (t+h/2, y+k2/2)
k4 = h * y' (t+h, y+k3)
y_new = (k1+2*k2+2*k3+k4) / 6