《天涯明月刀》引擎开发

安柏霖（frankan）

引擎架构师 Tencent

GAME DEVELOPERS CONFERENCE™ CHINA
SHANGHAI INTERNATIONAL CONVENTION CENTER
SHANGHAI, CHINA · OCTOBER 19–21, 2014

- OVERVIEW
- 场景
- 材质与光照
- 阴影
- 总结

- **OVERVIEW**
- 场景技术
- 材质与光照
- 阴影
- 总结

- 3D武侠MMORPG
- QuickSilver引擎，自研

场景广阔
全场景植被物件覆盖

丰富的天气

人物

良好的效率
23%的玩家使用的是intel集成显卡级别
剑试蔷薇

引擎团队

- **MMORPG**
  - 大，复杂，多人聚集
  - 开发周期长
- **中国市场**
  - 硬件配置分布广，低端很多
  - XP操作系统超过%50
- **中国开发团队**
  - 次世代经验相对较少

挑战 →

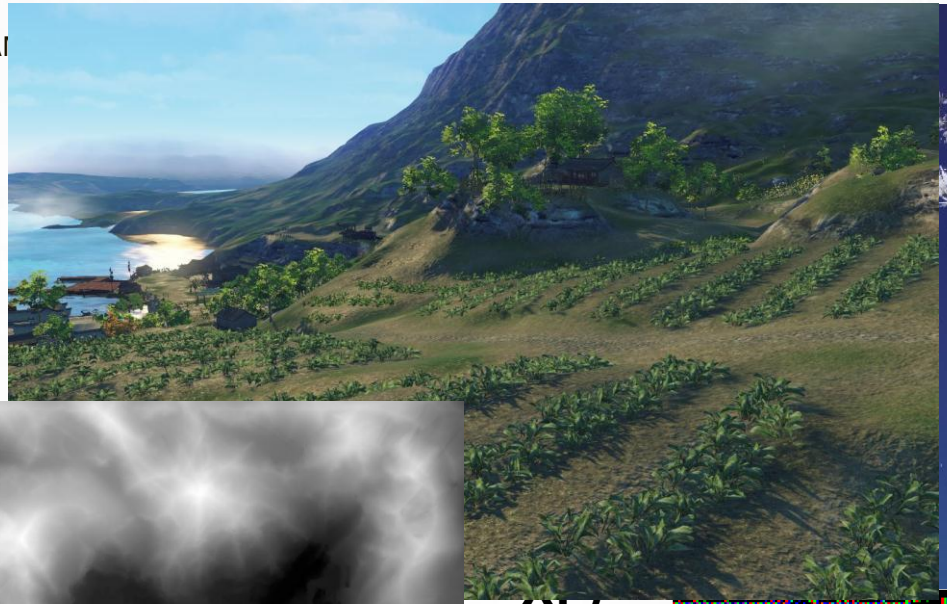
规模

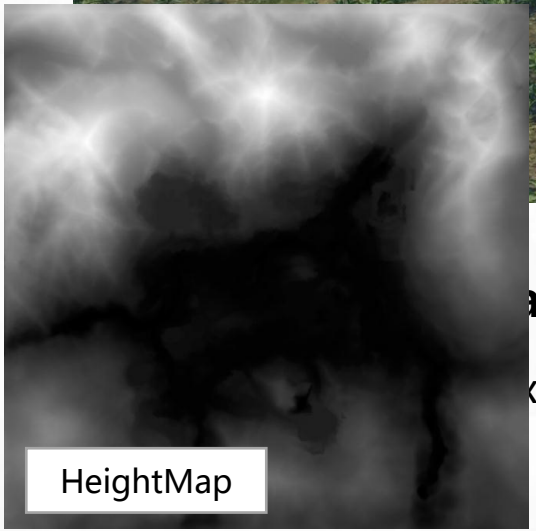
复杂度

效率

引擎开发实践

- **花更多的时间与美术协同工作**
  - 推进品质和效率
- **更倾向于简单，易于控制的技术实现**
  - 例如：固定材质 > 美术生成的节点材质
- **无处不在的数据监控**
- **花更多的时间用于优化**
  - Data Oriented Design, Cache Friendly, SSE…





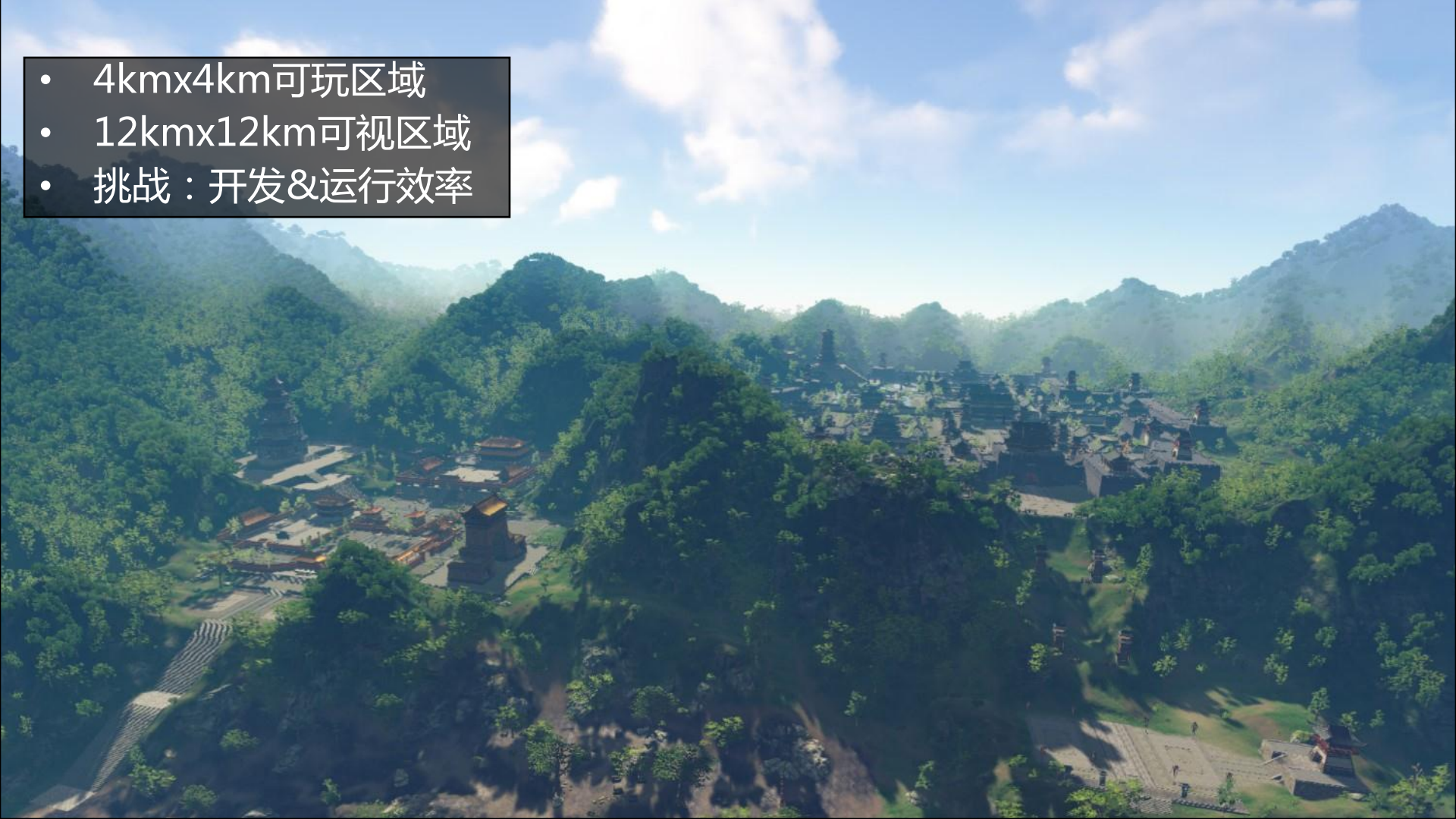
HeightMap

SkinningData

a G80架构的

xShader贴图读取能力

- OVERVIEW
- **场景**
- 材质与光照
- 阴影
- 总结


• 4kmx4km可玩区域
• 12kmx12km可视区域
• 挑战：开发&运行效率

运行效率

可视性检测

主线程：Entity级别的检测

渲染线程：node级别，各个pass的检测

LOD

Instancing

Geometry/Material/Billboard

Remapping

开发效率

工具pipeline

协作编辑

与外部工具互通

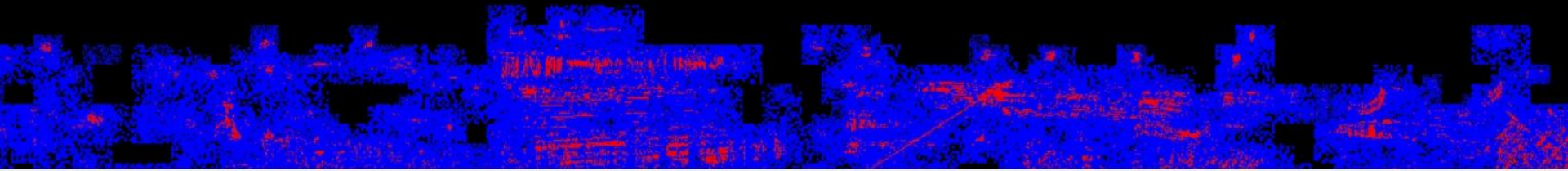
可视性检测

- **主线程 ： 跳过动画，粒子等计算**
  - Actor/Entity级别的检测
- **渲染线程 ： 跳过Draw Call**
  - 通用的可视性检测
    - 场景管理
    - 视锥体检测(frustum cull)
    - 软件光栅化剔除(software rasterization cull)
    - Contribution Cull（提出占屏幕空间小的物件）
  - 反射的可见性检测
  - 阴影的可视性检测

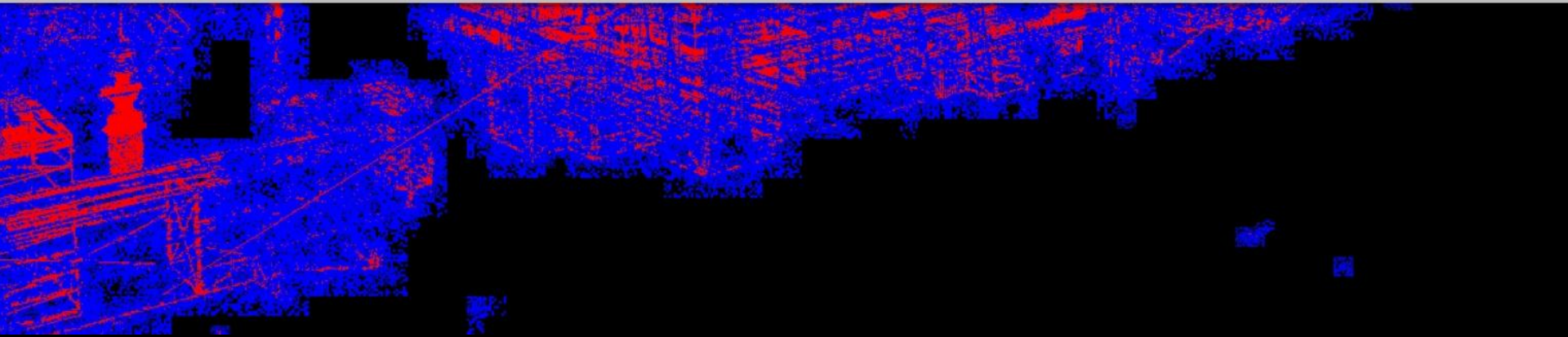
通用可视性检测

- **场景管理：两个层级的Cell/BVGrid—快速的内存访问**
- **SSE2指令集**
- **软件光栅化**
  - 地形
  - 内包围盒
- **Contribution Cull—剔除占屏幕非常小的物件**
  - 角色和普通物件需要分开，使用不同的阈值

软件光栅化



DrawCall：提升**47.8%**

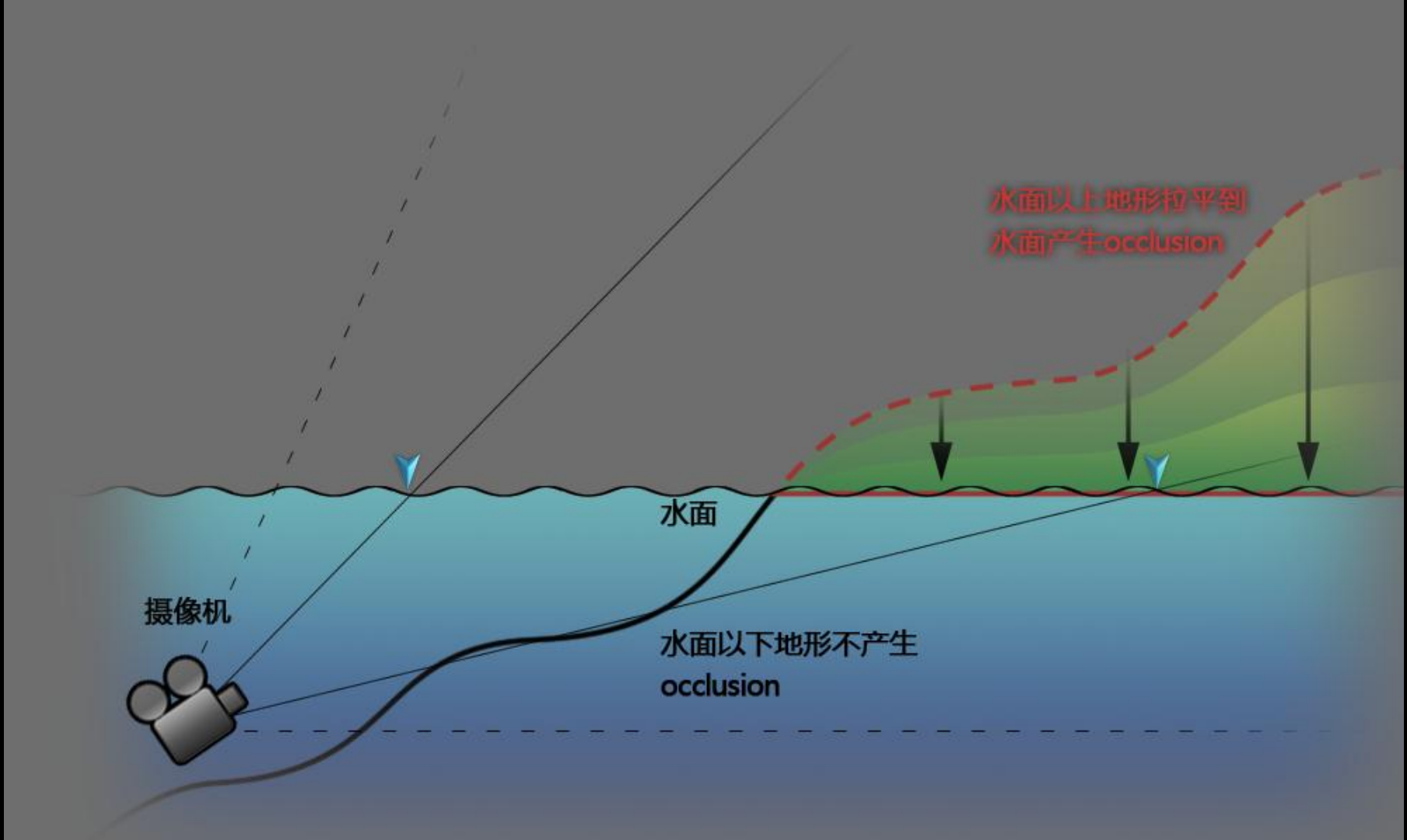


反射中的可视性检测

- **检测水是否可见**
  - 如果不可见，跳过反射渲染pass
  - 使用Occlusion Query判断水是否可见
- **会有较小的瑕疵**

反射中的可视性检测

- **摄像机的视锥可以根据水的大小进行缩减**
- **软件光栅化依旧会提升效率**
  - 要做一些调整
- **反射Mask**
  - 离线生成水以上的地形覆盖的Mask
  - 在VertexShader瓶颈的情况下就没用了
    - 简化模型，缩小反射Render Target


水面以上地形拉平到
水面产生occlusion
水面
摄像机
水面以下地形不产生
occlusion

反射中的可视性检测

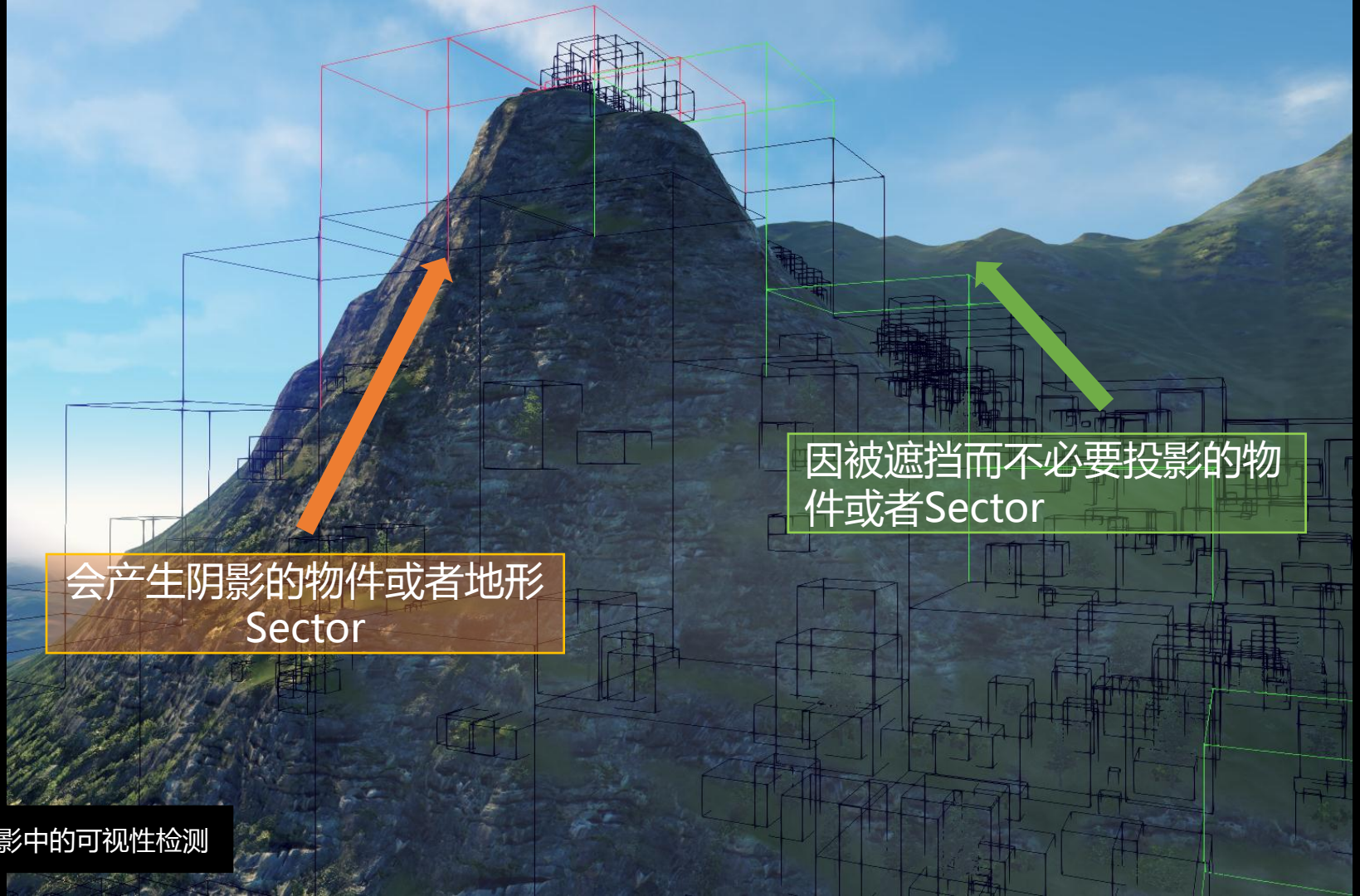
- **反射Mask**
  - 离线生成水以上的地形覆盖的Mask
  - 对PixelShader瓶颈的情况有用
    - 早期
  - 在VertexShader瓶颈的情况下就没用了
    - 简化模型，缩小反射Render Target

反射Mask：
地形覆盖的地方生成mask，
用于反射遮挡，
跳过Pixel Shader计算

阴影中的可视性检测

阴影的可视性检测

- **很大的计算量**
  - 2kmx2km的范围
  - 地形，静态物件和植被会产生投影
- **光的方向变化是固定的**
- **可以离线计算不需要投影的情况**
  - 已经被阴影覆盖的地形，静态物件和植被
  - 投影但是没有物体接受



阴影中的可视性检测

投影没有物体接收的地形
Sector可以不投影

阴影的可视性检测

- **离线进行**
- **将一天分成64时段，分别做visibility检测**
  - 每一个物件或者sector使用一个u64来存储
- **OcclusionQuery—被遮挡住的物件或者Sector**
- **StencilCount+OcclusionQuery—投影但没有东西接受的sector**
  - 使用D3DSTENCILOP_INCRSAT模式渲染
  - Query如果所覆盖的区域stencil都是<=1，就是投影不被接收

阴影的可视性检测

- **效率提升**
  - 静态物件和植被：节省超过13%
  - 地形sector：节省超过50%

平滑过渡

LOD

- **地形**
  - Geometry
    - 32x32->16x16->8x8
  - Material
    - DiffuseMap+NormalMap->DiffuseMap
- **物件**
  - 高精度模型->低精度模型->Billboard
  - 重新映射

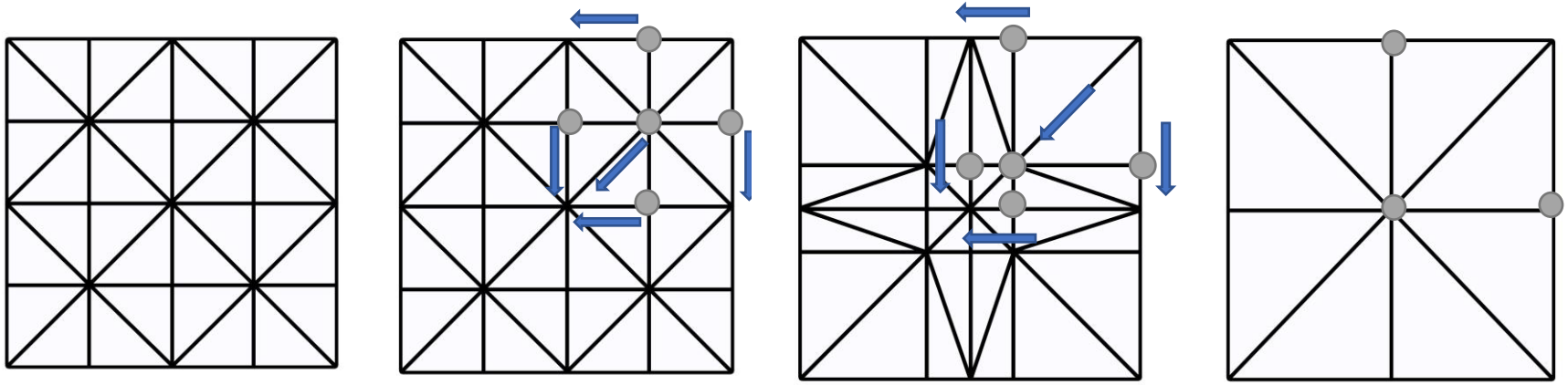
平滑过渡

Geometry Morph (CDLOD)

材质混合

Screen Door

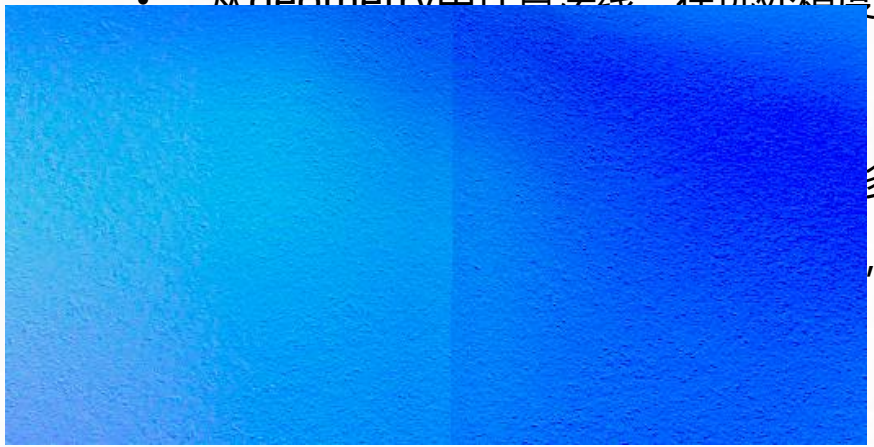
地形Geometry



Geometry Morph 的过程

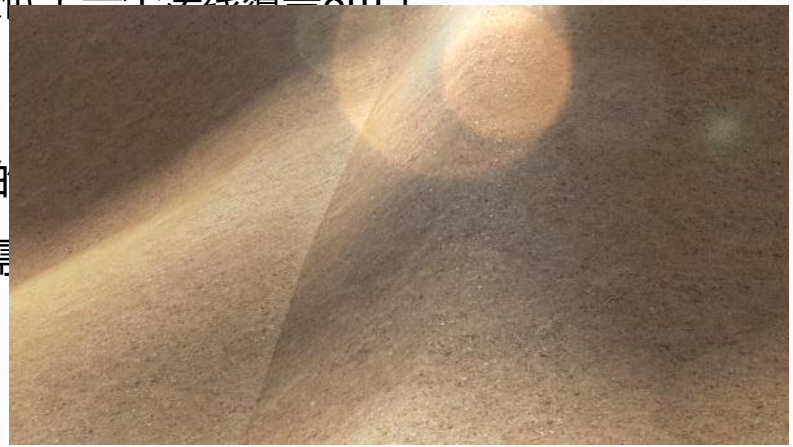
地形Geometry—实时计算法线（normal）的问题

- **法线精度：**
  - 从geometry中计算法线，在远处精度很低（一个法线覆盖8m）



多的

，需

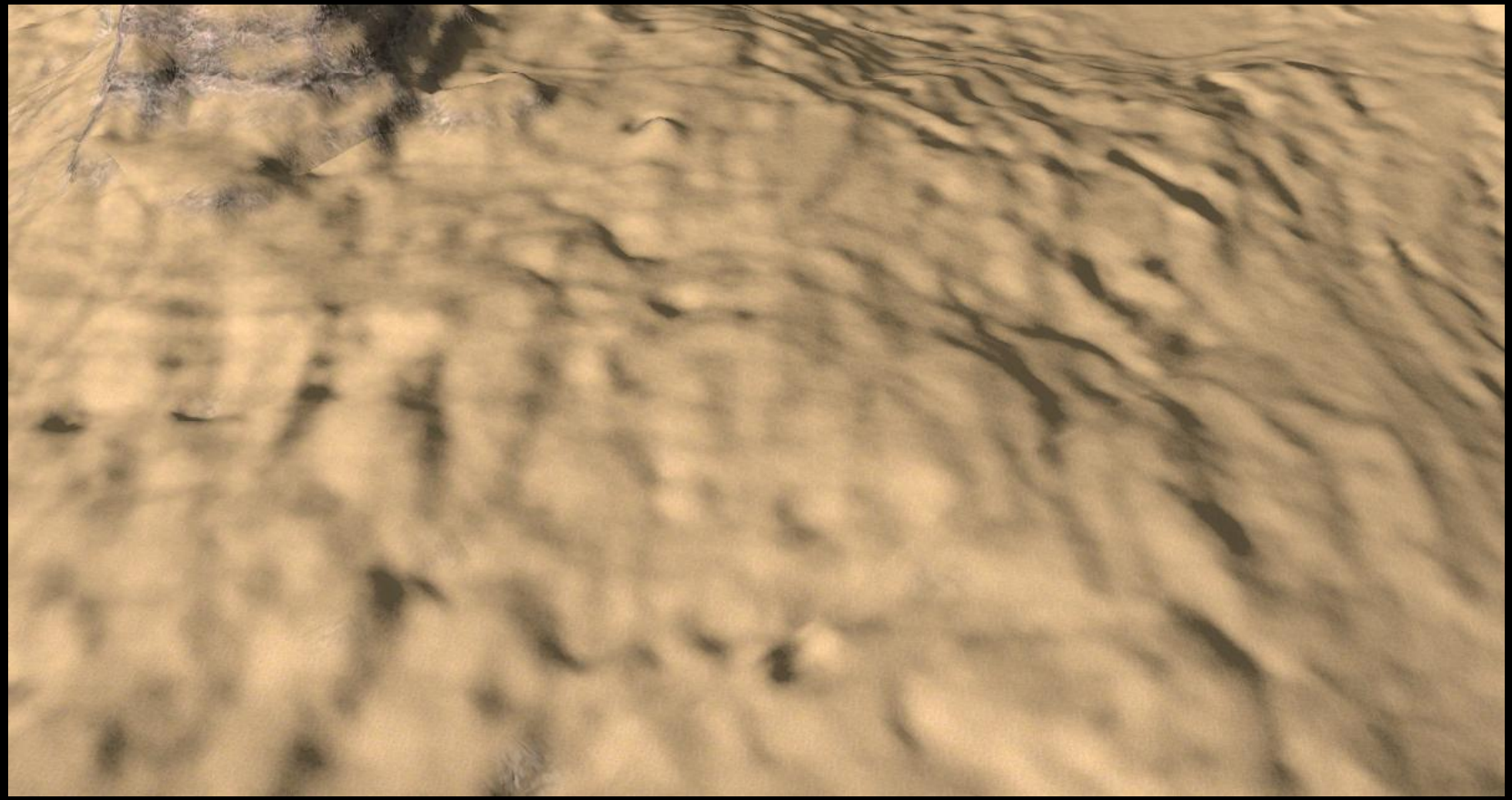


Tile：8x8个sector（256x256），一张高度图
不连续的Normal会造成视觉上的问题

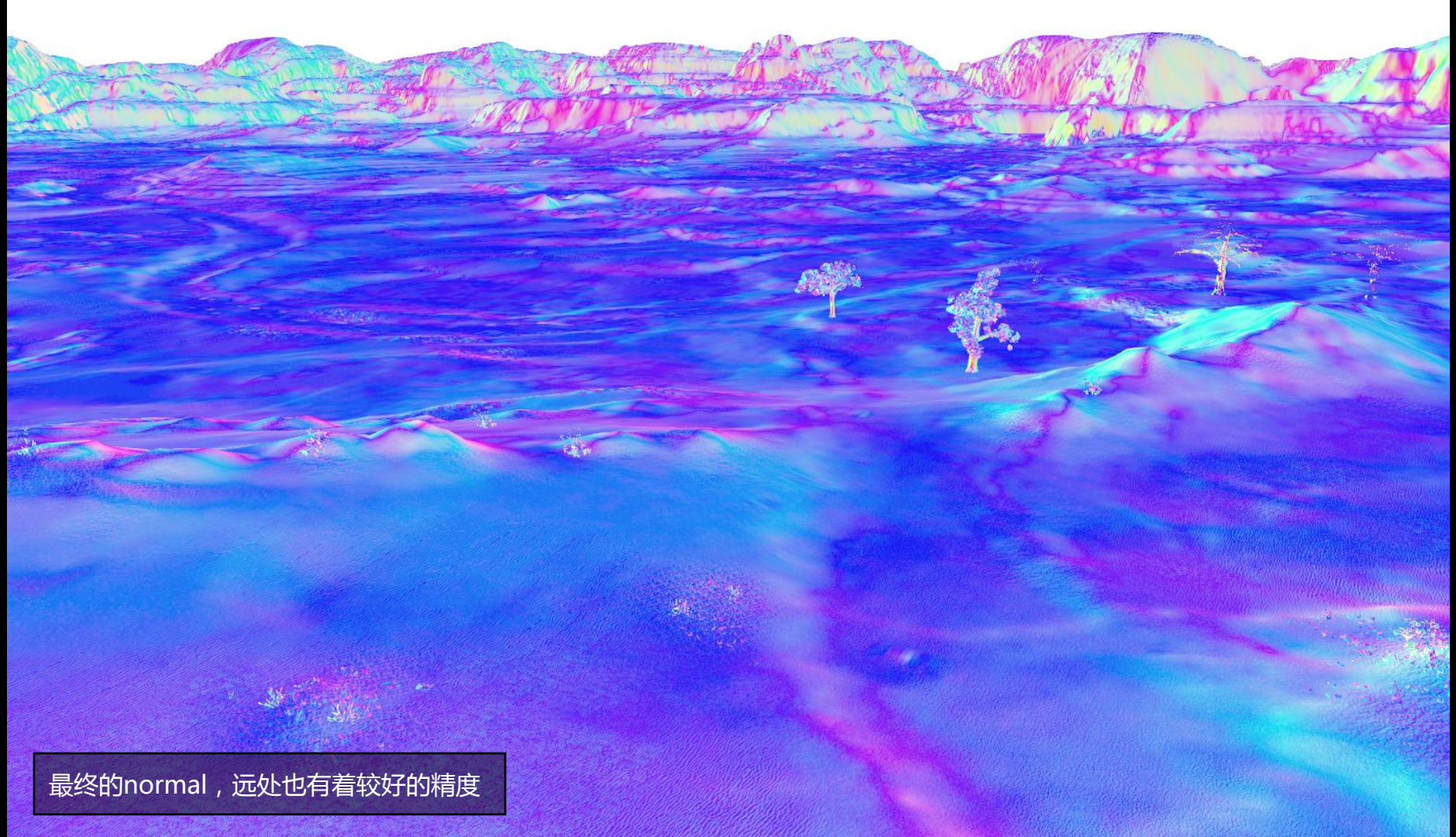
Tile级别的法线贴图

- 高精度的高度图（2m一格）产生的法线贴图
  - 256x256一个tile
  - 避免了实时计算以及接缝处理的消耗
  - 格式为Dxt5n：使用dxt5精度损失最小的A和G通道存储normal.xy
    - Dxt1压缩损失太大

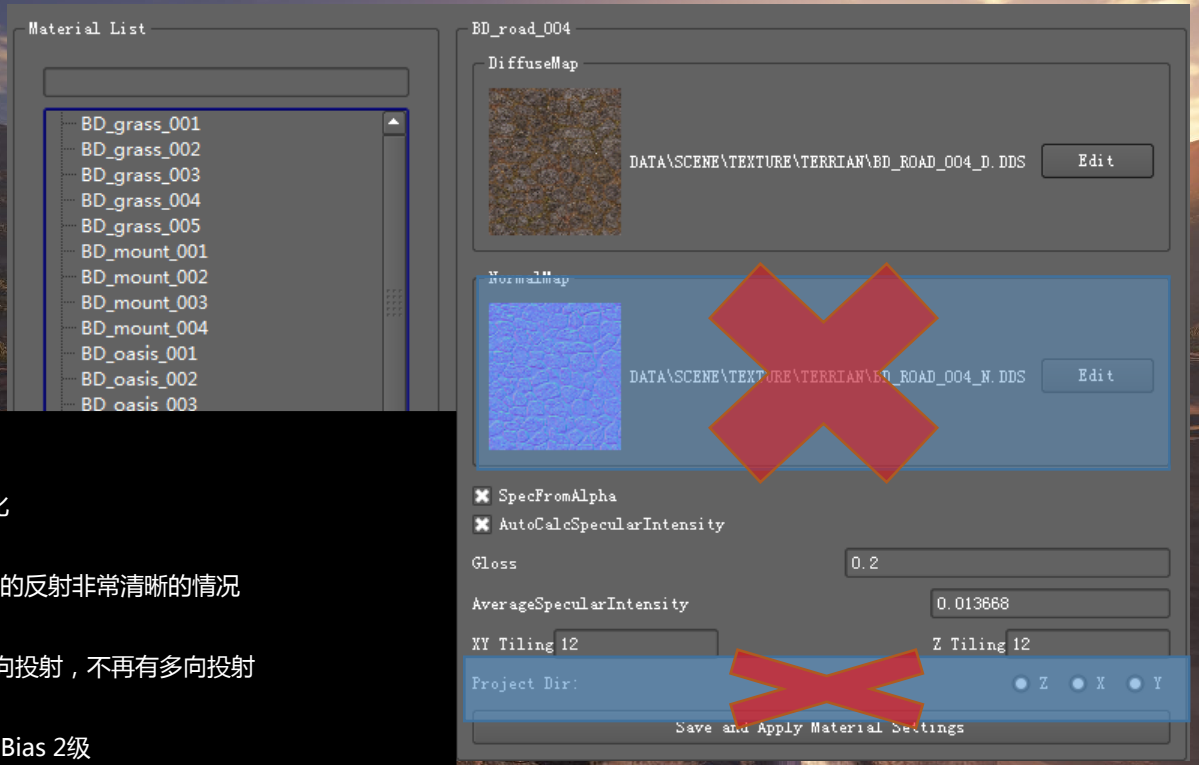
dxt1的Tile级normal



dxt5n的Tile级normal



最终的normal，远处也有着较好的精度



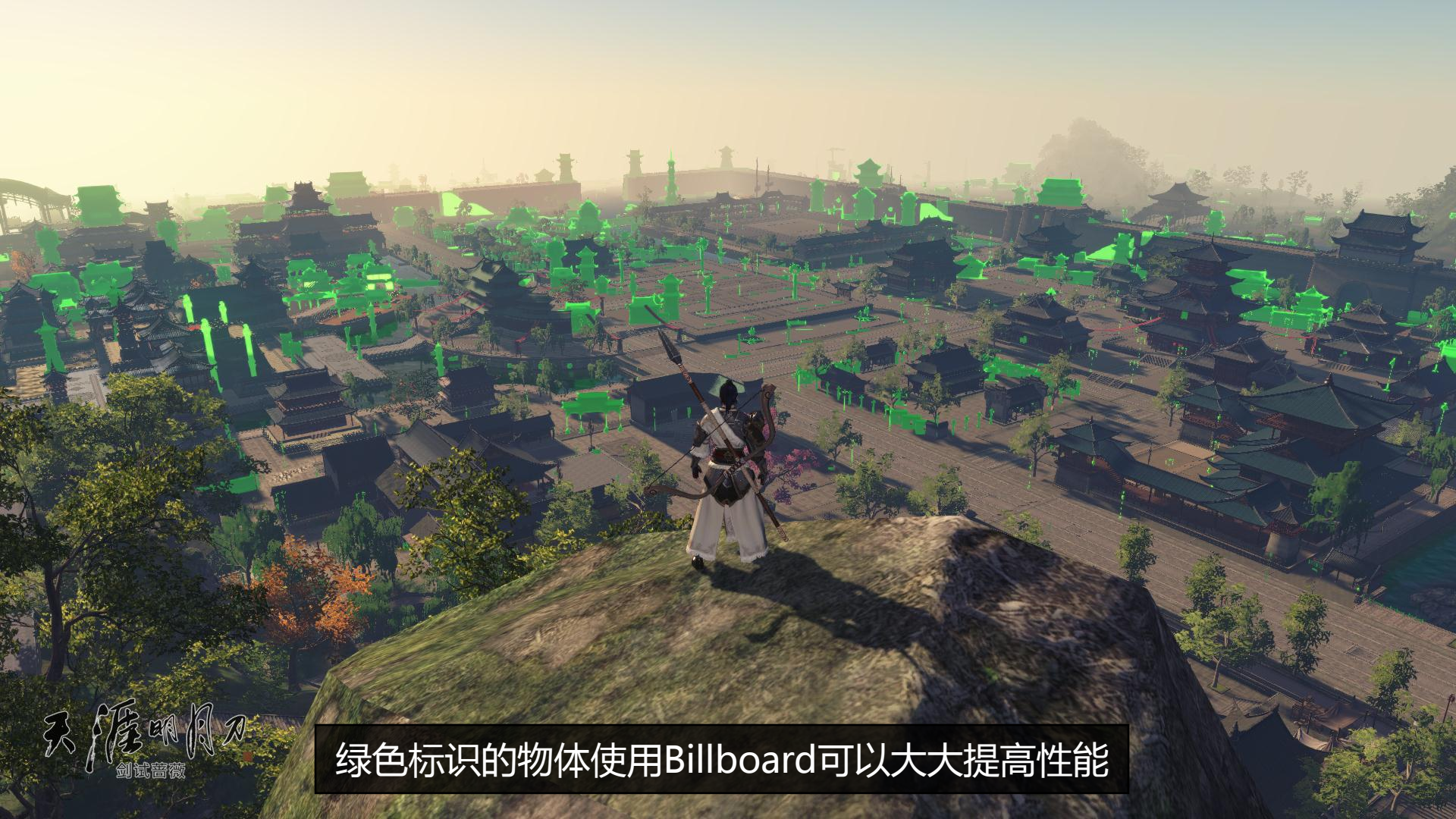
- 反射进一步简化
  - 避免水中的反射非常清晰的情况
  - 只在Z方向投射，不再有多向投射
  - MipmapBias 2级

材质LOD

反射中简化的地形材质

物件LOD

- **材质&模型**
  - 高精度模型&材质->低精度模型&材质->BillBoard
- **重新映射：人物&树**
  - 在种类太多的情况，内存和显存压力很大
  - 把一部分物件映射成其它mesh
    - 比如200个穿着不同玩家在一起的时候，重新映射成80种



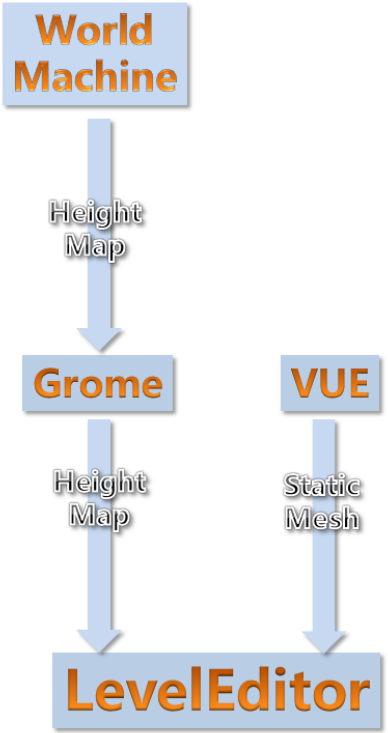
绿色标识的物体使用Billboard可以大大提高性能

Instancing

- **对多人，主城情况的渲染线程性能很重要**
- **动态进行Instancing判断**
  - MeshInstance在Mesh资源上有计数，大于10的Instance参与判断
    - 会剔除掉大部分Node，提升判断效率
  - 可见的同类Instance最后会进行Instancing
  - Pooled Instancing Buffer

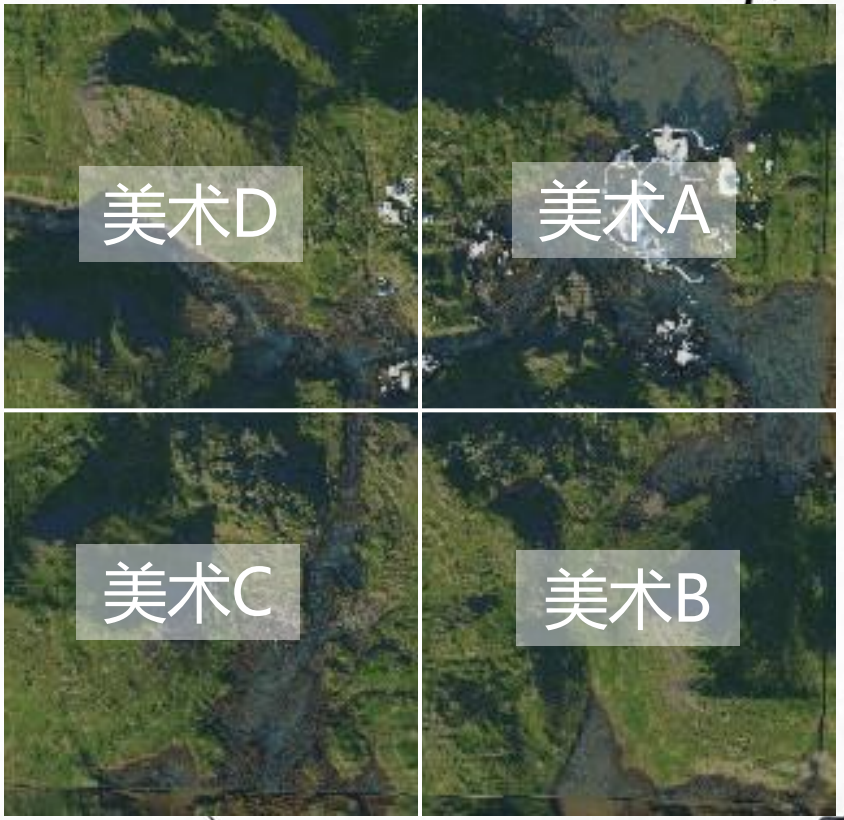
工具—导入导出

- 内部资源容易的和更专业的编辑工具互通
  - 高度图，材质等可以方便的使用VUE，Grome，Photoshop
  - 程序&美术开发效率都更好



工具—导入导出

- 协作编辑
  - 多人合作一张地图
  - 导入导出
    - 高度图，材质
    - Entity等逻辑数据

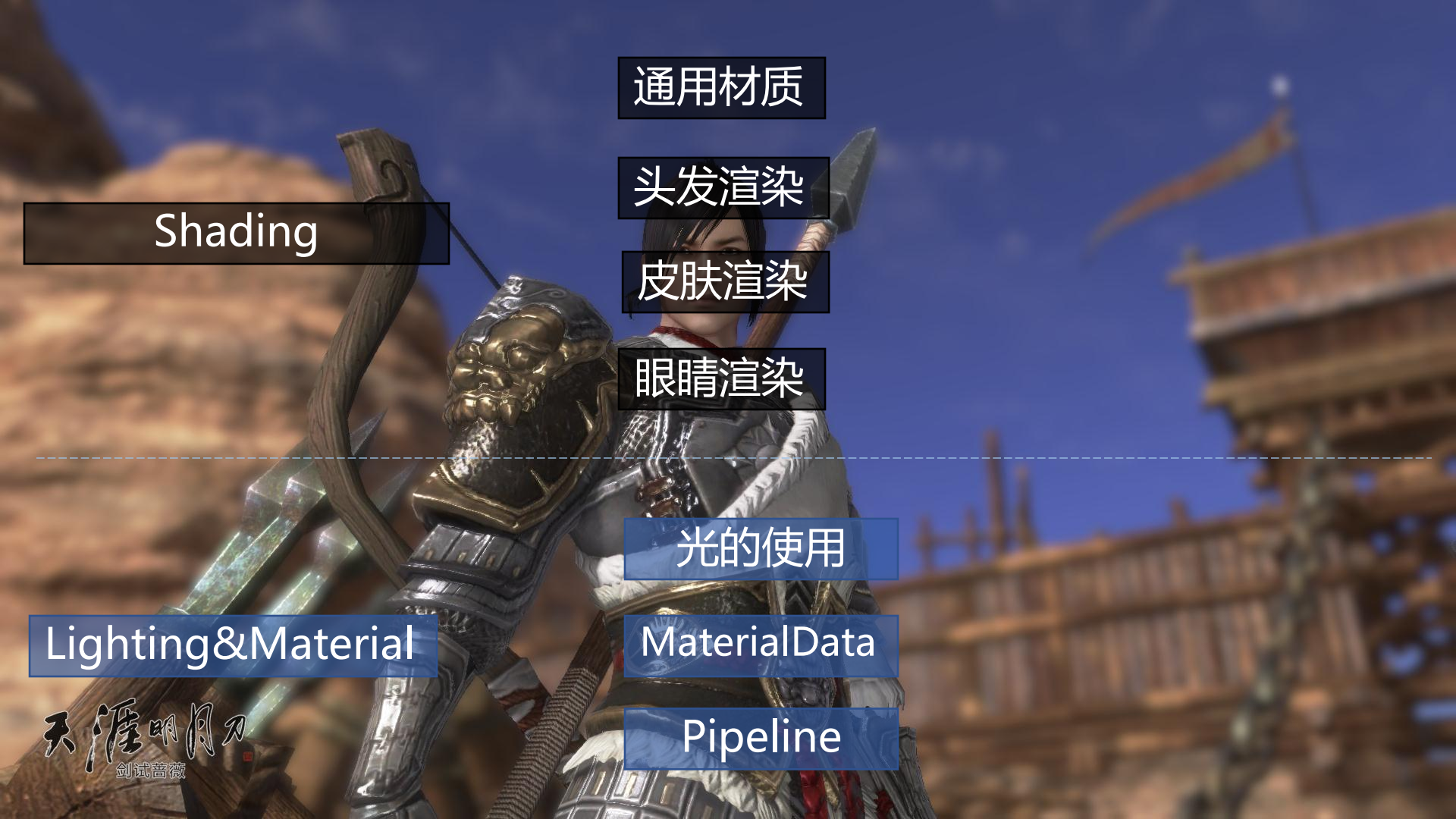


浮点数精度

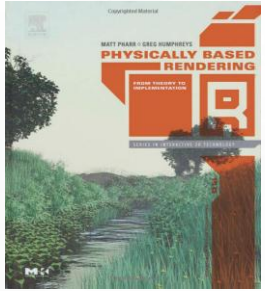
- 坐标数值很大导致精度不足
  - 尤其是经过旋转计算之后
  - 出现抖动，不贴合等问题
- 所有矩阵运算需要考虑精度
  - 比如：将Translation转到CameraLocal坐标系中，然后进行矩阵操作

```
float4 worldToCam = float4(posWorld.xyz-cameraPos.xyz,1.0f);
float4 homoPos = mul(worldToCam, PartialViewProj);
```

- OVERVIEW
- 场景
- **材质与光照**
- 阴影
- 总结


通用材质
头发渲染
Shading
皮肤渲染
眼睛渲染
光的使用
Lighting&Material
MaterialData
Pipeline
天涯明月刀
剑试蔷薇

基于物理的渲染



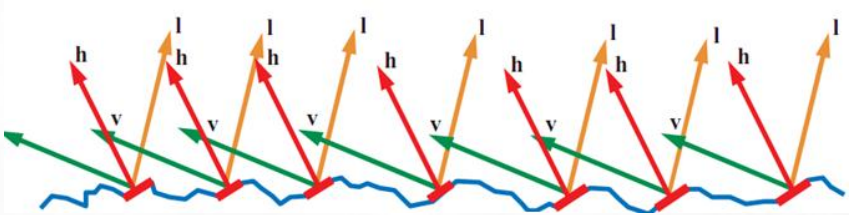
SIGGRAPH 2010 Course

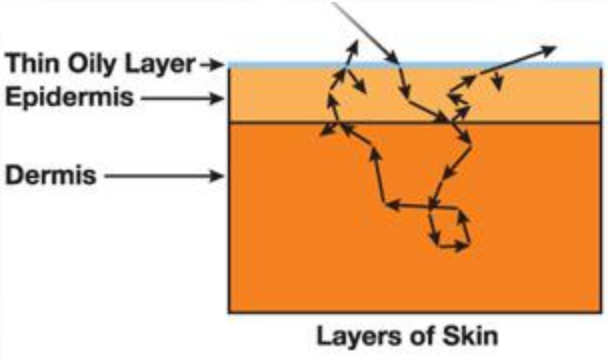
SIGGRAPH&GDC...
在大量的游戏中流行起来

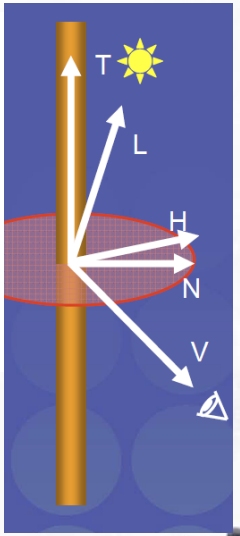
《天涯明月刀》光照材质在2011年定下，
收到tri-Ace和Treyarch的启发

基于物理的渲染

- 根据光在材质上传播物理特性
  - 数学建模
  - 简化
  - 渲染







基于物理的渲染

- 源自真实的数学模型
  - 真实的光照效果
  - 多变的光照环境中保持光照效果正确
  - 美术需要调的参数更少更简单
- 对《天涯明月刀》格外的重要
- 《天涯明月刀》根据需要做了一些选择和调整

写实风格

更简单的美术制作流程

通用材质

FinalLighting

=

DiffuseLighting

+

SpecularLighting

Specular Lighting

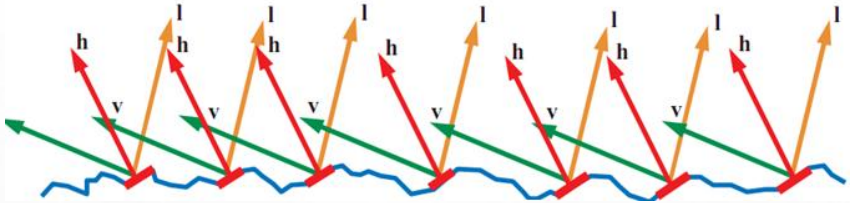
=

主Specular Lighting

+

次Specular Lighting

主Specular Lighting



MacrofacetBRDF

$$\frac{F(\mathbf{l},\mathbf{h})G(\mathbf{l},\mathbf{v},\mathbf{h})D(\mathbf{h})}{4(\underline{\mathbf{n}\cdot\mathbf{l}})(\underline{\mathbf{n}\cdot\mathbf{v}})}$$

主Specular Lighting

$$\frac{F(\mathbf{l},\mathbf{h})G(\mathbf{l},\mathbf{v},\mathbf{h})D(\mathbf{h})}{4(\underline{\mathbf{n}\cdot\mathbf{l}})(\underline{\mathbf{n}\cdot\mathbf{v}})}$$

MacrofacetBRDF

tri-Ace的模型：

D : blinn-phong

F : Schlick’s approximation

G : Neumann-Neumann GAF

$$(0.0397436n+0.0856832)\frac{F_{spec}(f_0)(N\cdot H)^n}{\max(N\cdot L,N\cdot E)}.$$

FinalFunction

次Specular Lighting

- Image Based Lighting:
  - 离线烘培EnvMap
  - CubeMapGen prefilter生成平滑的CubeMap
- GlossyReflection:
  - texCUBElod(EnvMap, float4(reflectDir, mipNum*(1.0f-gloss)))

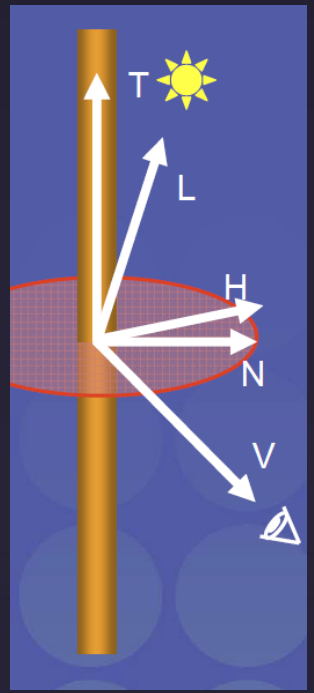
次Specular Lighting的环境贴图

- 128x128
- LDR
- 美术根据天气，时段配置scale参数
- 次Specular Lighting只用于角色
  - 没有reflection occlusion
  - 会有较多瑕疵
- 角色和场景使用不同的光照参数
  - 后面会具体介绍细节
  - 烘培环境贴图和使用环境贴图在不同光照环境下
  - 美术来根据天气，时段配置环境贴图的scale值更加的简单直接

没有reflection occlusion，
次Specular Lighting在普通物件上会有较多瑕疵

头发渲染

Kajiya-Kay




```
float tdh = dot(tangentVec, halfVec);
Spec = pow(sqrt(1.0f-tdh*tdh), specPower);
```

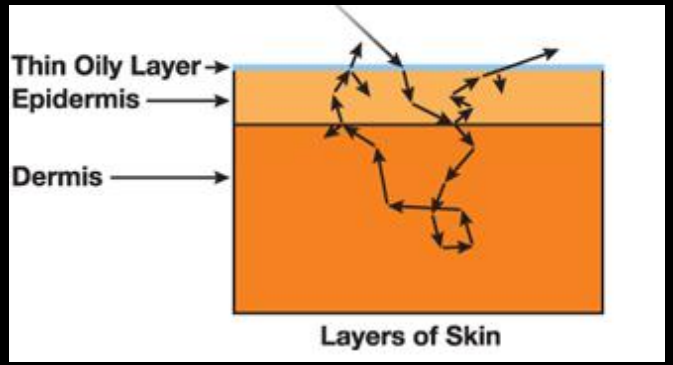
平滑边缘：

- 渲染深度Pass
  - AlphaTest掉Alpha<1的部分
  - 防止大部分乱序
- 渲染透明Pass
- 少数乱序问题通过数据制作避免这种情况的出现

皮肤渲染

皮肤渲染

- Specular Lighting：
  - Kelemen/Szirmay-Kalos BRDF
- Diffuse Lighting
  - 5S(ScreenSpaceSubSurfaceScattering)
  - 效率非常稳定
  - 质量不错
    - 没有TextureSpace质量高



使用stencil跳过皮肤以外的区域

在LightingPass的最开始，首先渲染皮肤的DiffuseLighting

Blur的时候如果取到的是黑色，则丢弃

这样就可以防止皮肤以外的color混到皮肤的光照中来

进行SubSurfaceScattering

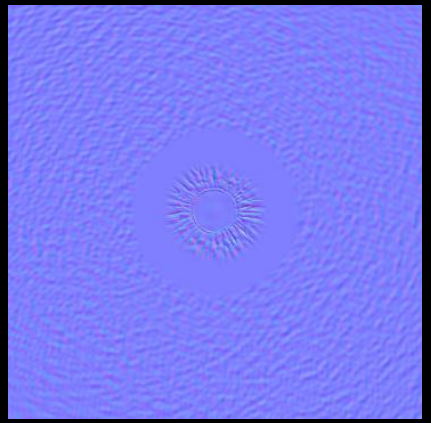
ApplyDiffuseColor
计算Specular

眼睛渲染

眼睛渲染

- 对于细腻的情绪表达非常重要
  - 早期使用贴图，表现力差很多
- [Jimenez12]给出了很好的方案，
- 由于视距的关系，我们可以大幅度简化，以降低资源消耗
- Reflection Occlusion由于有眨眼的关系，不能简单使用Occlusion Map，需要额外处理

眼睛渲染



不是一个平滑的球，使用一个normal map来达到效果

[jimenez12]中的GlossMap等可以不使用

眼睛渲染

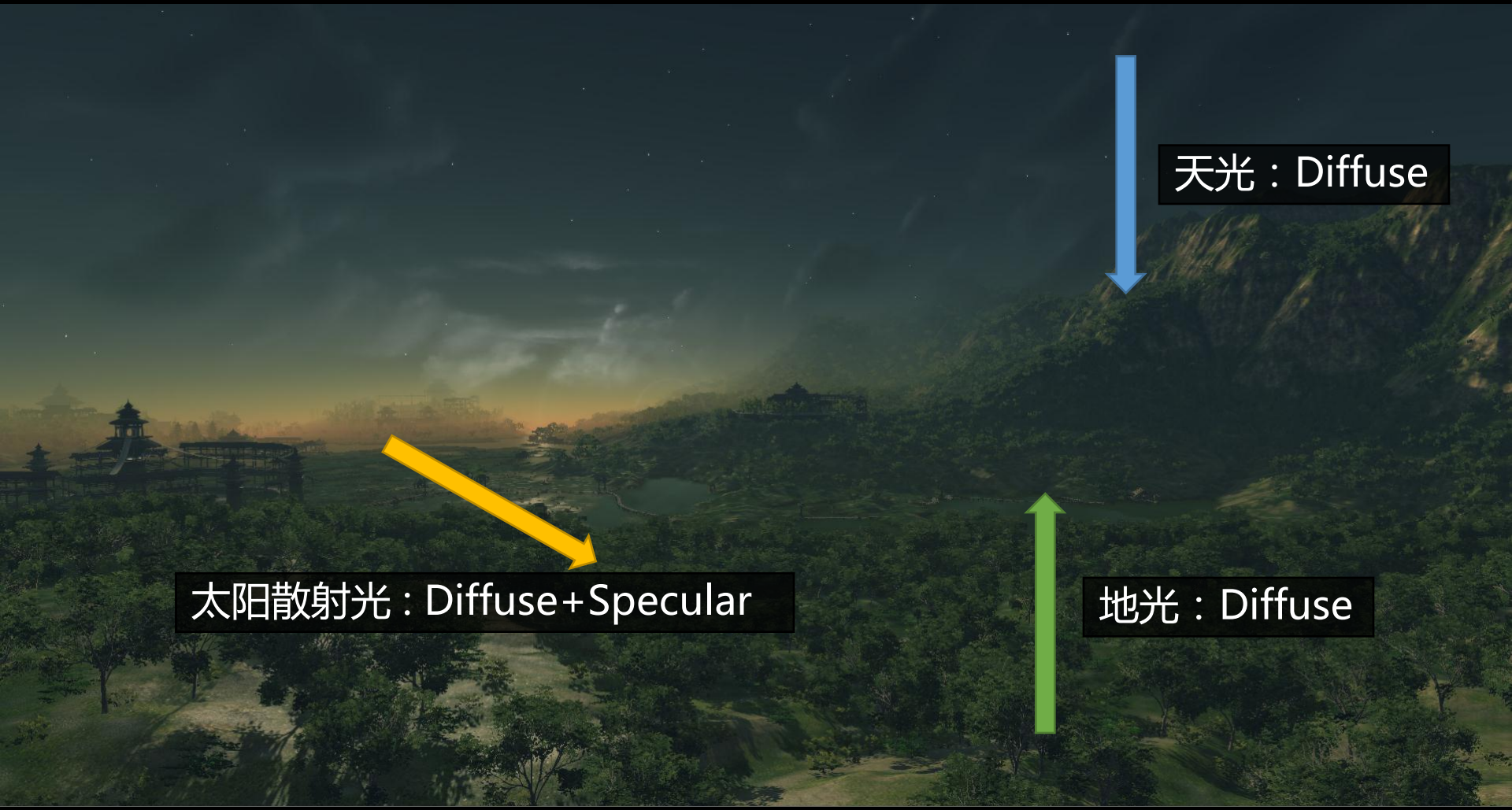
睫毛使用头发的渲染方式，有透明pass

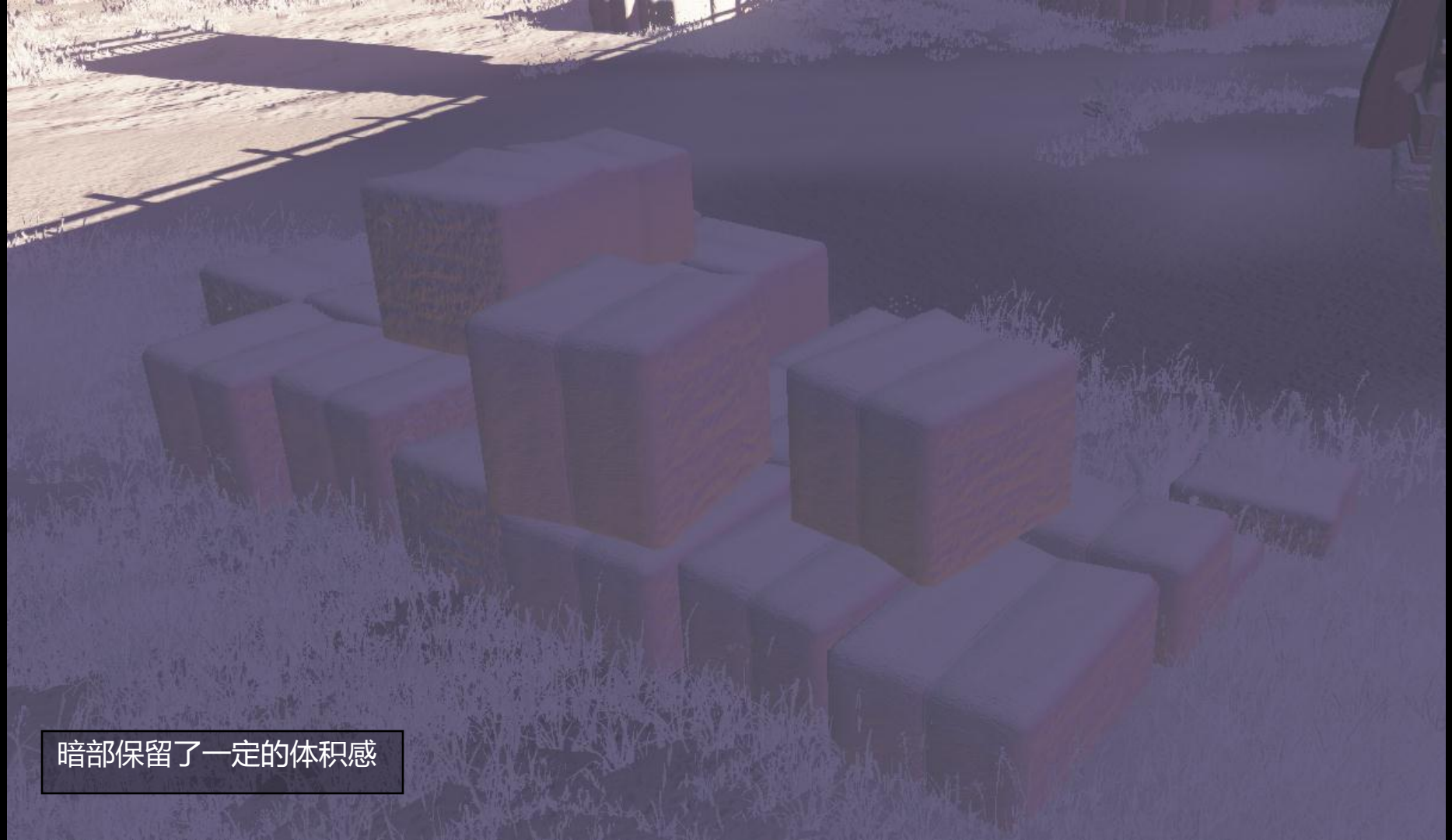
在睫毛上加一个覆盖在眼球边缘的透明层，达到Lighting的occlusion的效果

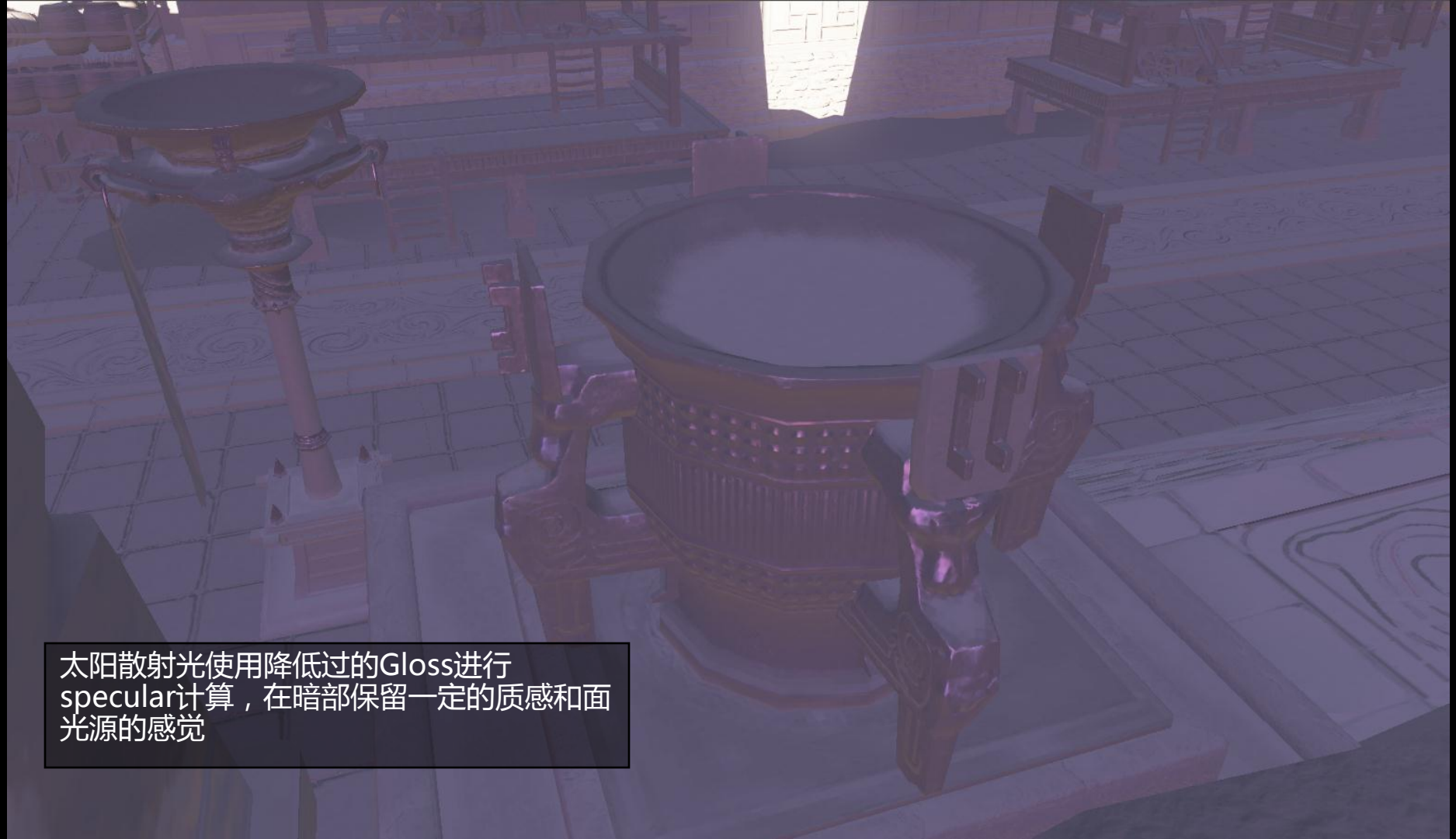
ReflectionOcclusionMap在有眨眼动作的时候就不适用了

Ambient Lighting

- 场景没有Secondary Specular Lighting
  - 让暗部的立体感更有挑战
- 使用3个半球光作为AmbientLighting
  - Sky, Ground, SunScatter


天光：Diffuse
太阳散射光：Diffuse+Specular
地光：Diffuse


暗部保留了一定的体积感



太阳散射光使用降低过的Gloss进行specular计算，在暗部保留一定的质感和面光源的感觉

G

3/5 材质光照

角色光照

- 需要任何情况下看起来都很棒
- 光照参数和场景分开，由角色美术单独调试
- 渲染是两个pass，不同参数，使用stencil区分
- 增加摄像机灯光—Camera View方向的一个光照
- 头发在阴影中DirectLighting保留%30到%40

摄像机灯光

很多玩家的截图都是背光的

MaterialData

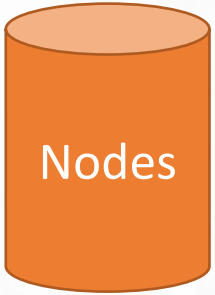
- 固定材质
  - 根据需求，由程序员实现，暴露参数
  - 运行效率高
  - 易于控制种类，复杂度
  - 易于开发
  - 相对来说MMORPG比单机游戏更有理由使用固定材质

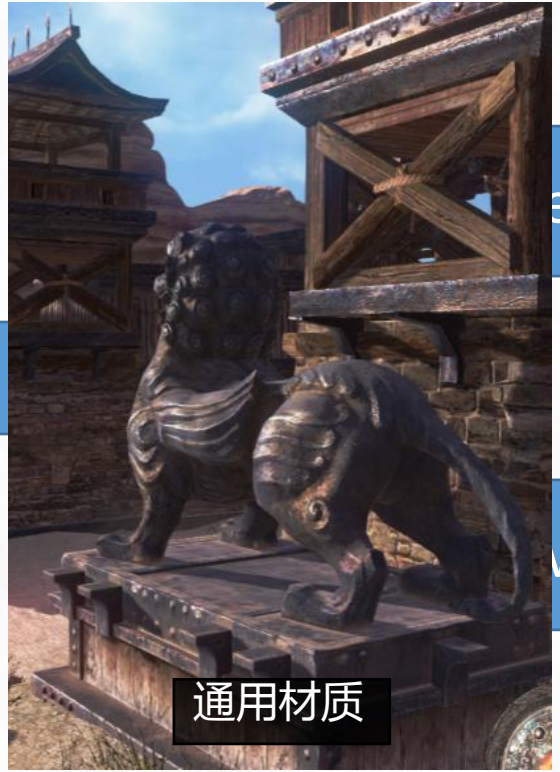
MaterialData

- 大部分的材质：尽可能的节省显存
  - Diffuse Map : Dxt1/Dxt5
  - Normal Gloss Map : Dxt1
    - RG—Normal的x y : 在shader中计算Z
    - B ： Gloss
    - 没有处理SpecularAliasing
  - Specular Map : Dxt1

3/5 材质光照

LightingPipeline : Deferred/Forward混合模式





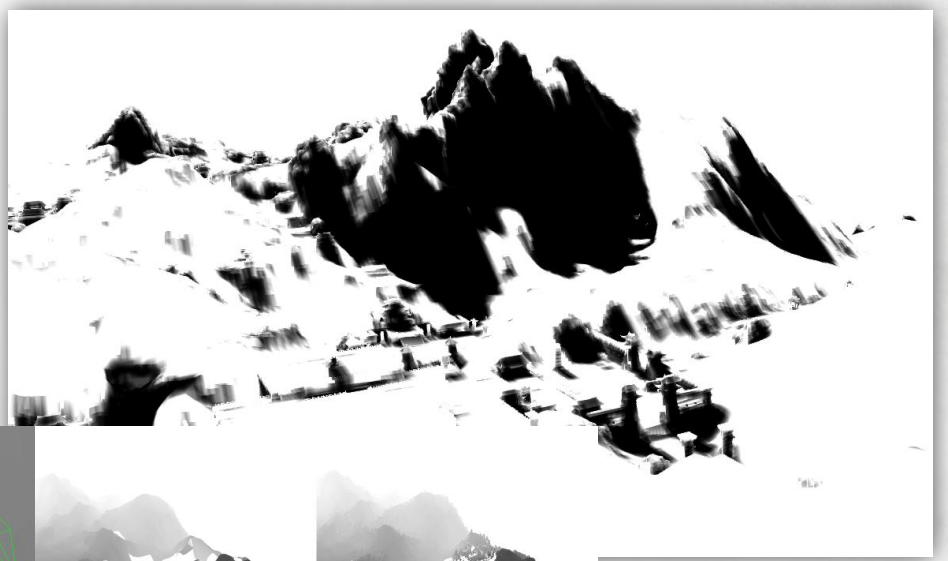
同时进行计数准备
Instancing

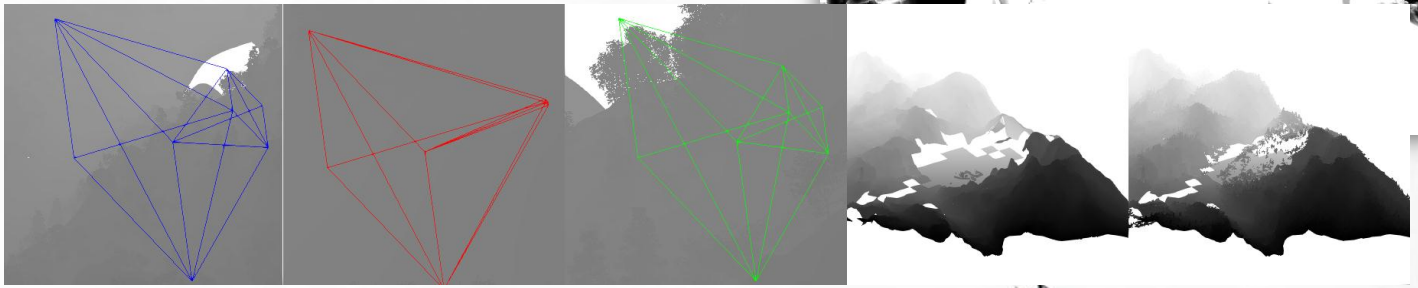
- OVERVIEW
- 场景
- 材质与光照
- **阴影**
- 总结

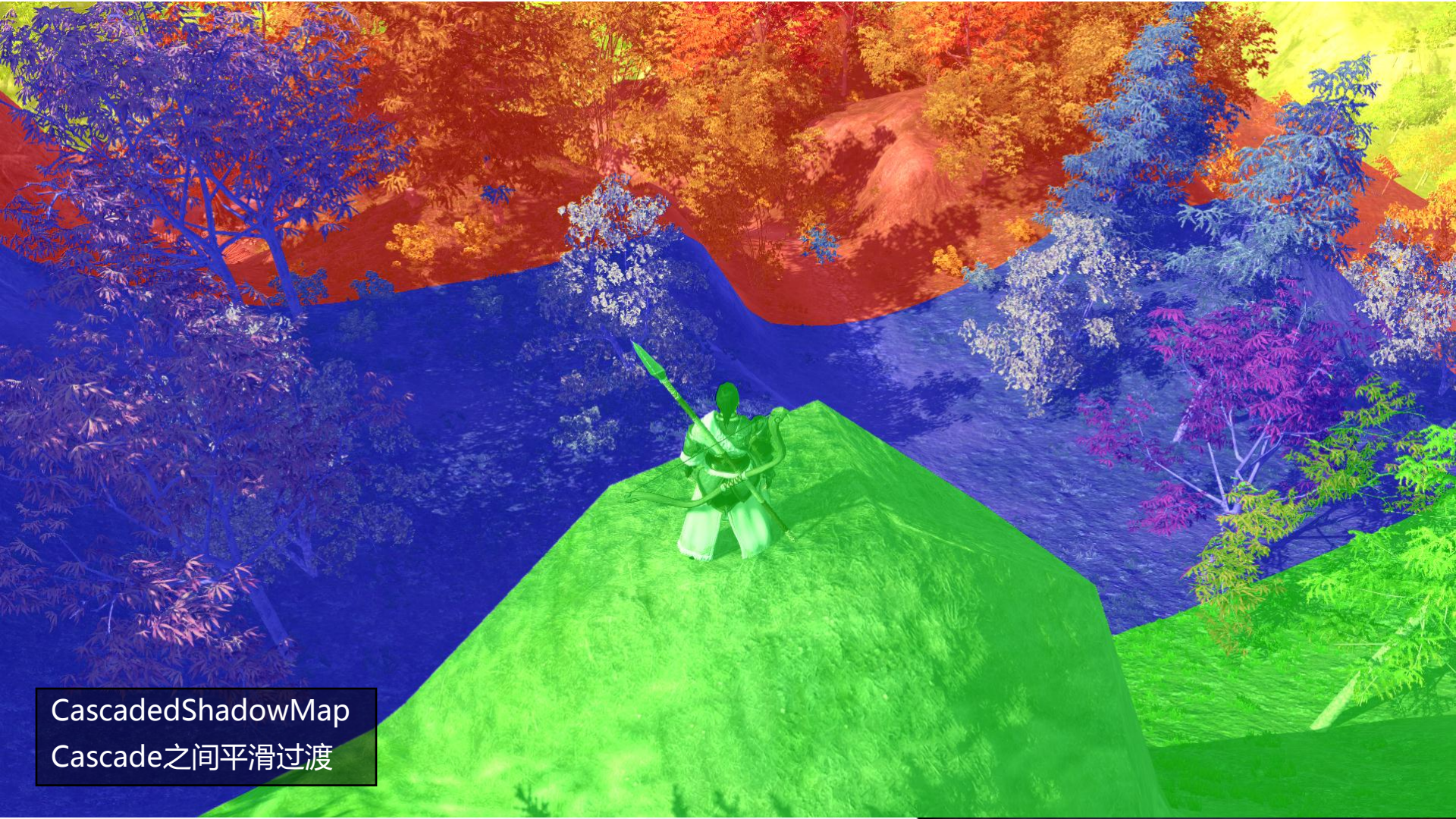
Shadow

阴影--CascadedShadowMap

- 4个cascade，平滑过渡
- Pack到一个LightViewBuffer中
- 前3个Cascade的每帧更新，覆盖大约200米
- 第4个cascade覆盖超过2km

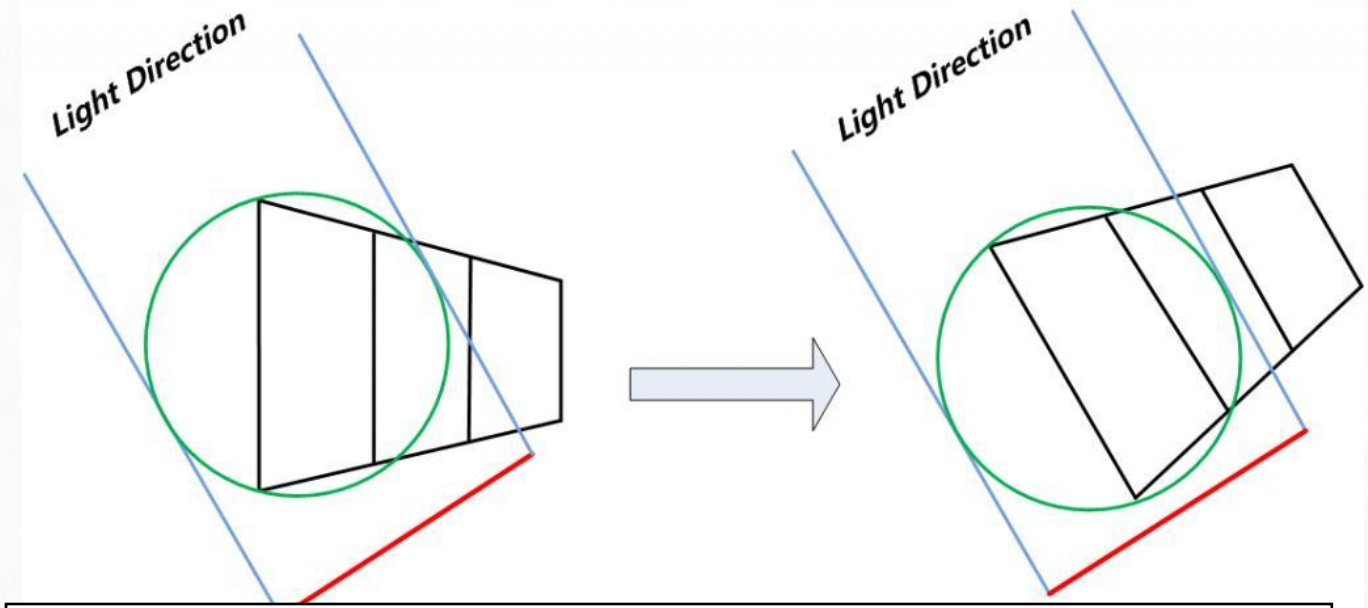





CascadedShadowMap
Cascade之间平滑过渡

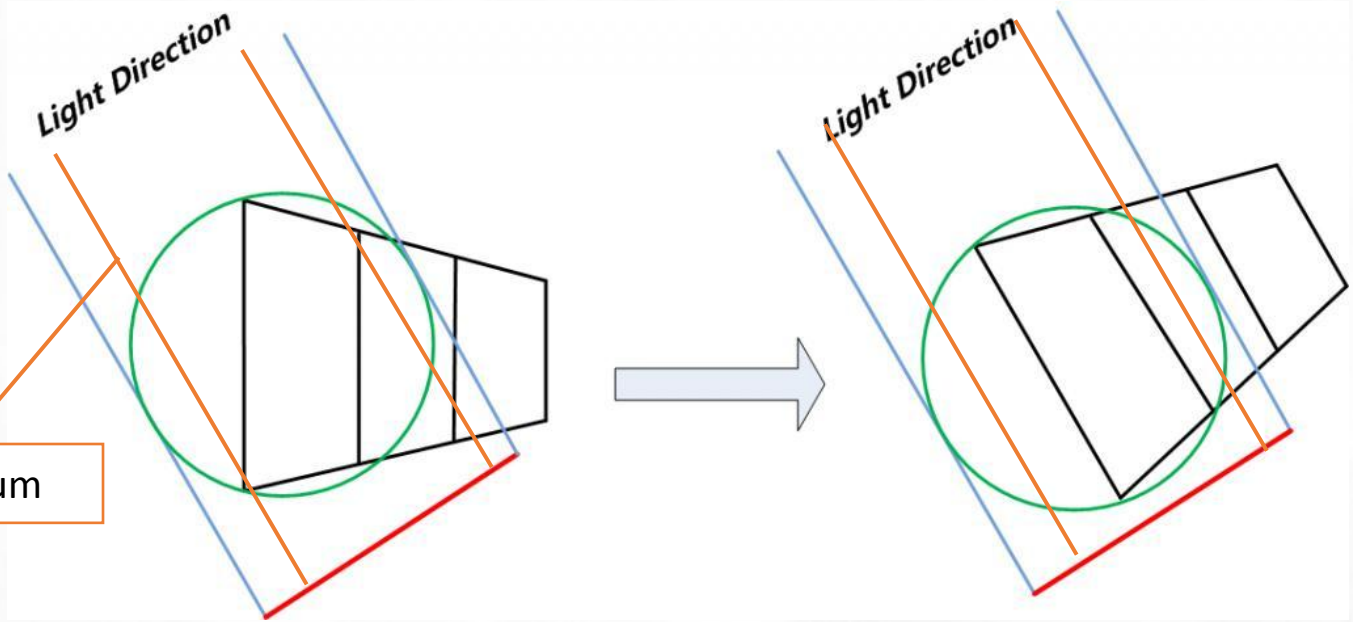
通过Blend来平滑过渡
避免了生硬的cascade之间的过渡（也消耗了更多性能）
让更高的ShadowMap利用成为可能
CascadedShadowMap
Cascade之间平滑过渡

更高的ShadowMap利用率



没有平滑过渡的话，为了保证完整性，Frustum需要完整包含Cascade的包围球
而Frustum最外侧部分只是为了很小部分cascade椎体

更高的ShadowMap利用率

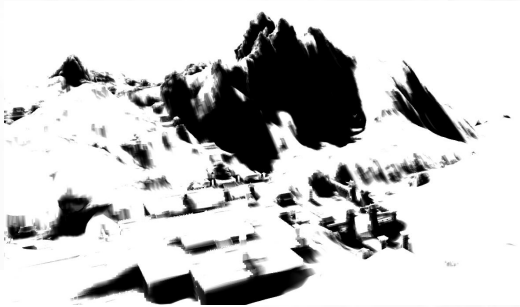


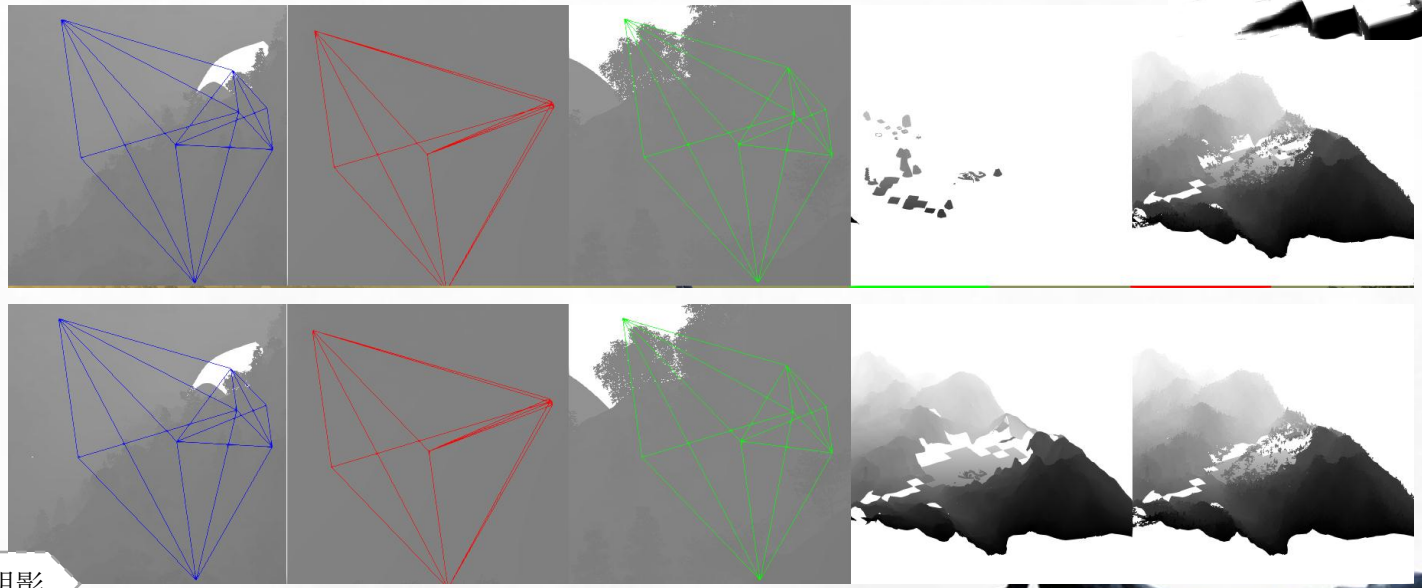
新frustum

有了平滑过渡，可以把Frutum缩小到原来的0.85x0.85=0.72，少部分覆盖不到的地方有下一个cascade混合处理

第4级阴影

- 覆盖超过2km
- 分帧更新，每20秒一切换（随光的方向，位置的移动）
- 切换过程中，平滑过渡





ShadowCache

- 可以**节省约%70**的static mesh&terrain sector的shadow中渲染
- 用在第三个Cascade里
- 在第1，2两个cascade中意义不大，值不回显存“票价”

LightViewDepth

移动过程中，大部分深度信息可以重用

只渲染没有cache的部分

Shadow其他

- Snapping处理抖动
- Shadow Mask : 对ShadowMap构建pass**达到%10到%60**的优化

- OVERVIEW
- 场景
- 材质与光照
- 阴影
- **总结**

The End