可微渲染

《differentiable Visual Computing》读书报告

姓名：陈嘉博

学号：22251158

学院：软件学院

专业：软件工程

目录

1 可微渲染简介 4
2 可微渲染原理 5
2.1 从数学本质理解可微渲染 5
2.2 从宏观视角理解可微渲染 12
3 可微渲染应用 15
4 可微渲染展望 17
局部最小值、过参数化和预处理 17
完全可微分的计算机图形学 17
用于其他计算的自动微分处理器 17
5 小结 18
参考文献 19

简　介

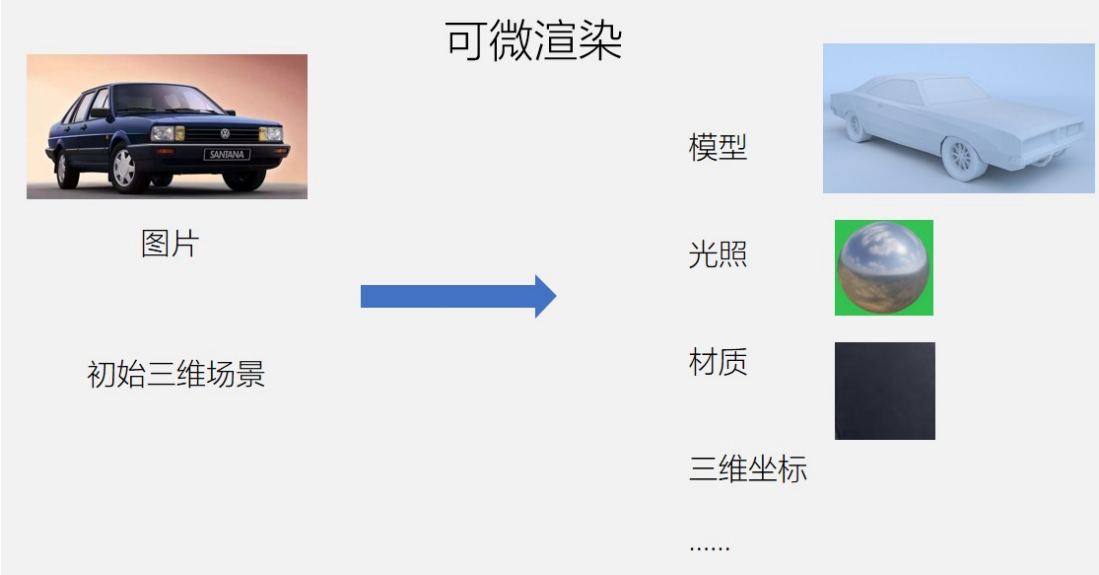
本报告分析的论文是2020年度SIGGRAPH**最佳博士论文《differentiable Visual Computing》**，作者是**Tzu-Mao Li**博士，来自**麻省理工学院**。在颁奖词中，SIGGRAPH称他的博士论文为**新兴的可微计算机图形学奠定了基础**，李子懋是**物理可微渲染领域的先行者**。

本报告在该博士论文的基础上加入了近年来可微渲染的其他研究成果，以便读者能对可微渲染领域的最新研究进展有更全面的认识。

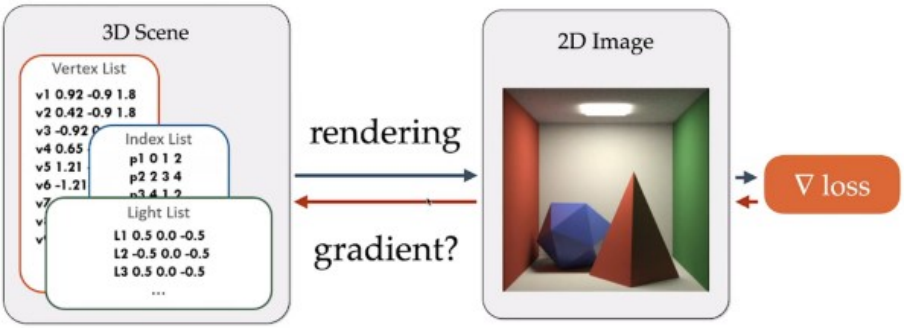
依靠传统方法通过美术工作者的经验生成包括模型、材质等在内大量的数据资产显然费时费力，对构建宏大虚拟世界构成了挑战。而近年来的可微渲染技术为我们提供了一个可靠的、能够依靠数据驱动的方法生成大量数据资产的工具，这个工具让游戏世界的构建变得更加方便。

1 可微渲染简介

近年来，随着图形硬件以及渲染技术和深度学习技术的快速发展，作为二者之间的桥梁，可微渲染受到了学界的广泛关注。在计算机图形学中，要使计算机能够展现一张人脸，需要根据场景光照、人脸模型、人脸材质等相关参数，将人脸的三维图形转化为用计算机显示器的栅格形式表示的二维图片，简单来说，就是通过 3D 场景获得 2D 图像，这一过程被称为渲染。



可微渲染可以实现渲染的逆过程，即逆向渲染，也就是通过 2D 图像获得 3D 场景。可微渲染使用的方法是计算渲染过程中的导数以实现将渲染过程放入神经网络中以解决更复杂的问题。该方法可以通过一张或几张二维人脸图片，获得图片所展示场景的光照、人脸模型、人脸材质等参数。人脸材质主要包括肤色、纹理、界面反射参数等内容，界面反射参数包括粗糙度和高光反射率，加入界面反射参数可以更好地描绘计算机中的三维物体。



2 可微渲染原理

2.1 从数学本质理解可微渲染

分析计算机图形和图像处理算法的导数是可微渲染的关键。我们将使用它们来最小化成本函数，解决逆问题，并指导采样过程。直观地说，函数的导数表征了给定点的局部行为，例如，如果我将点移动到这个方向，输出值会变大还是变小？这允许我们找到导致某些函数值的点，例如最大化效用函数，或最小化输出和目标之间的差异。

如果想对自动微分有一个更加全面的了解，本文推荐阅读 Griewank 和 Walther 的教材[19]。

有限差分与符号导数

在讨论自动微分算法之前，回顾生成导数的其他方法，并将它们与自动微分进行比较是很有用的。

导数的一种常见近似如下，有时称为数值导数。给定一个函数 f(x)和一个输入 x，我们通过少量 h 扰动 x 来近似导数：

$$\frac{df(x)}{dx} \approx \frac{f(x+h)-f(x)}{h} \quad \text{or}$$
$$\frac{df(x)}{dx} \approx \frac{f(x+h)-f(x-h)}{2h}.$$

这种近似的问题有两个方面。首先，计算机系统中步长 h 的最佳选择取决于问题。如果步长太小，则宣誓点表示的舍入误差太大。另一方面，如果步长太大，则结果与真导数的近似值很差。第二，该方法不适用于多元函数。对于具有 100 个变量和标量输出的函数，计算全梯度向量需要对原始函数进行至少 101 次求值。

另一种选择是将函数 f 的内容视为一系列数学运算，并象征性地表示函数。事实上，大多数解释规则都是机械的，我们可以应用这些规则来生成 f(x)。然而，在我们的例子中，f(x)通常是一种算法，符号定义并不能很好地与符号的数量成比例。考虑以下代码：

```
function f(x):
    result = x
    for i = 1 to 8:
        result = exp(result)
    return result
```

图 2-1：一个迭代计算嵌套指数的代码示例，用于演示符号表示和自动表示之间的区别。

```
function d_f(x):
    result = x
    d_result = 1
    for i = 1 to 8:
        result = exp(result)
        d_result = d_result * result
    return d_result
```

上面的代码输出与符号导数完全相同的值（方程 2.3），但明显更有效，这是由于自动定义更好地使用中间值和对常见子表达式的仔细分解。

生成导数的算法

为了更好地理解自动微分，在介绍全自动解决方案之前，我们将首先讨论如何手动微分代码示例。我们从只有函数调用和基本运算（如加法和乘法）的程序开始。特别是，我们不允许递归或循环函数调用。在第 2.3.1 章的后面，我们概括了处理循环和分支等控制流以及处理递归的思想。在本章中，我们假设所有函数调用都是无边的。据作者所知，目前还没有已知的自动识别算法来转换带有边的任意函数。

自动识别的关键是链式规则。考虑输入 x 和输出 z 的以下代码：

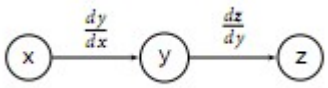
```
y = f(x)
z = g(y)
```

假设我们已经知道导数函数 d f(x) dx 和 dgfiyfi dy，并且我们对输出 z 相对于输入 x 的导数感兴趣。我们可以通过应用链式法则来计算导数：

```
dydx = dfdx(x)
dzdy = dgdy(y)
dzdx = dzdy * dydx
```

我们可以递归地应用规则来生成导数函数，直到该函数是一个我们知道解析导数的基本函数，例如加法、乘法、sin（）或 exp（）。

一个有用的计算模型是计算图。它可以用于表示变量之间的依赖关系_图的 e 节点是变量，边是相邻顶点之间的导数。在上述情况下，图表是线性的：



计算计算图的导数涉及图的遍历，以及连接输入和输出的不同路径的集合。

在实践中，大多数函数都是多变量的，有时我们希望有多个导数，例如梯度向量。在这种情况下，不同的导数在计算图中可能有共同的路径，这些路径可以被分解，这会极大地影响效率。考虑以下代码示例及其计算图：

```
y = f(x0, x1)
z = g(y)
w0, w1 = h(z)
```

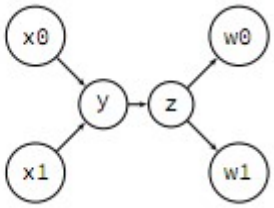
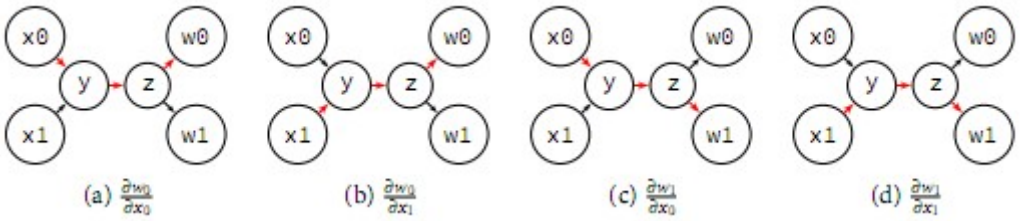


图 2-2：具有两个输入 x0、x1 和两个输出 w0、w1 的代码示例和计算图

两个输出和两个输入之间有四个导数。我们可以通过遍历计算图中的四条对应路径来获得它们：



例如，在（a）中，w0 相对于 x0 的导数是三个红色边的乘积：

$$\frac{\partial w_0}{\partial x_0} = \frac{\partial w_0}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial x_0},$$

并且在（b）中，w0 相对于 x1 的导数为

$$\frac{\partial w_0}{\partial x_1} = \frac{\partial w_0}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial x_1}.$$

我们可以观察到，一些导数在计算图中共享公共子路径。例如，fiw0 fix0 和 fiw0 fix1 以上的两个导数共享相同的子路径 y，z，w0。因此，我们可以对这两个导数的子路径和预乘 fiw0 fiy fiw0 fi z fiz fiy 进行因子化。在更大的计算图中，这种因子分解可能会对导数代码的性能产生巨大影响，甚至会影响输入或输出数量方面的时间复杂性。

不同的自动微分算法在计算题中使用不同的方法找因子。最极端的情况下，找到导致最小运算的因式分解是 NP 难问题 154。幸运的是，在许多常见情况下，例如梯度向量的因子分解，都有有效的解决方案。

如果输入是标量变量，无论输出中有多少变量，前向模式自动迭代都会生成与原始算法具有相同时间复杂度的派生代码。另一方面，如果输出是标量变量，无论有多少输入变量，反向模式自动迭代都会生成与原始算法具有相同时间复杂度的派生代码_后一种情况特别有趣，因为这意味着我们可以以相同的时间复杂度计算梯度（廉价梯度原理），这对于各种优化和采样算法都很有用。

接下来，我们将演示几种计算导数的算法，同时仔细考虑常见的子表达式。我们展示了如何使用控件 ow、循环或递归将数字算法转换为生成导数的代码。

前向传播

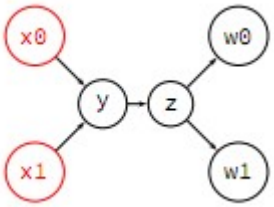
我们从最简单的算法开始，通常称为前向模式自动迭代，有时也称为对偶数。前向遍历从输入到输出的计算图，计算中间节点相对于沿途所有输入变量的导数。当输入维度较低而输出维度较高时，前向模式是有效的，因为对于计算图中的每个节点，我们需要计算每个输入变量的导数。

在计算机图形学中，正向模式已用于计算 3D 渲染 90、118 中的纹理预滤波的屏幕空间导数，用于计算物理模拟 72 的微分方程中的导数，以及用于估计镜面物体 243 中的运动。前向模式对于计算黑森矩阵也很有用，可以先应用前向模式，然后在每个输出上应用反向模式，以获得完整的黑森矩阵。

我们将使用图 2-2 中的前一个示例描述正向模式。从输入开始，目标是使用链式规则传播关于输入的导数。为了处理函数调用，对于输出变量引用的每个函数 f（x），我们生成一个导数函数 df（x，dx），其中 dx 是 x 相对于输入变量的导数。

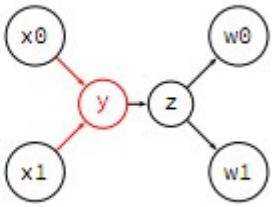
我们从输入 x0、x1 开始，生成 $\frac{\partial x_0}{\partial x_0} = 1$ 和 $\frac{\partial x_1}{\partial x_1} = 1$。我们使用 2D 向量 dx0dx 来表示 x0 相对于 x0 和 x1 的导数。

```
dx0dx = {1, 0}
dx1dx = {0, 1}
y = f(x0, x1)
z = g(y)
w0, w1 = h(z)
```



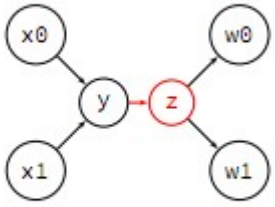
然后，我们获得 y 相对于输入的导数。我们假设我们已经对 f 应用了前向模式自动微分，所以我们有一个导数函数 df（x0，dx0dx，x1，dx1dx）。

```
dx0dx = {1, 0}
dx1dx = {0, 1}
y = f(x0, x1)
dydx = df(x0, dx0dx,
          x1, dx1dx)
z = g(y)
w0, w1 = h(z)
```



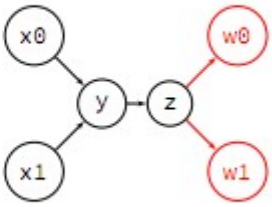
然后我们将导数传播到 z：

```
dx0dx = {1, 0}
dx1dx = {0, 1}
y = f(x0, x1)
dydx = df(x0, dx0dx,
          x1, dx1dx)
z = g(y)
dzdx = dg(y, dydx)
w0, w1 = h(z)
```



最后，我们将 z 的导数传播到输出 w0、w1。

```
dx0dx = {1, 0}
dx1dx = {0, 1}
y = f(x0, x1)
dydx = df(x0, dx0dx,
          x1, dx1dx)
z = g(y)
dzdx = dg(y, dydx)
w0, w1 = h(z)
dw0dx, dw1dx = dh(z, dzdx)
```



前向模式自动迭代生成的代码的时间复杂度是原始算法时间复杂度乘 O(d)，其中 d 是输入变量的数量。它对于输入变量少的函数是有效的。

然而，对于导数的许多应用，我们需要用数千甚至数百万个输入变量来定义函数。使用正向模式将是不可行的，因为我们需要计算计算图中每个输出的所有输入变量的导数。幸运的是，还有另一种称为反向模式自动迭代的算法，它可以在只有单个输出时生成与原始算法具有相同时间复杂度的派生代码，而不管输入变量的数量。

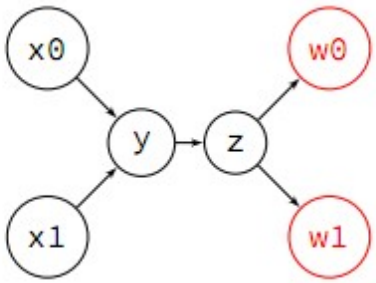
反向传播

反向模式将导数从输出传播到输入，而正向模式则将导数从输入传播到输出。对于计算图中的每个节点，我们计算所有输出相对于该节点处变量的导数_当输入维数大而输出维数低时，前反向模式更有效。然而，反向模式的实现也更加复杂，因为它需要反向运行原始算法来传播导数。

我们再次使用图 2-2 中的前一个示例来说明反向模式的工作原理。与转发模式类似，我们需要处理函数调用。对于输出变量引用的每个函数 yf（x），我们生成一个导数函数 df（x，dy），其中 dy 是最终输出相对于函数 s 输出 y 的导数向量（相反，在正向模式下，导数函数将输入导数作为自变量）。

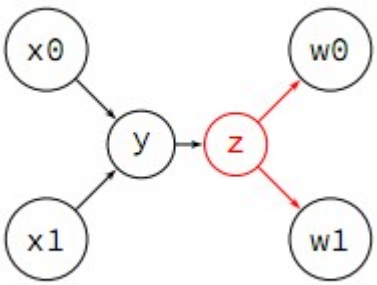
我们使用 f(w0)和 f(w1)从输出 w0、w1 开始。我们使用 2D 向量 dwdw0 来表示 w0、w1 相对于 w0 的导数。

```
y = f(x0, x1)
z = g(y)
w0, w1 = h(z)

dwdw0 = {1, 0}
dwdw1 = {0, 1}
```



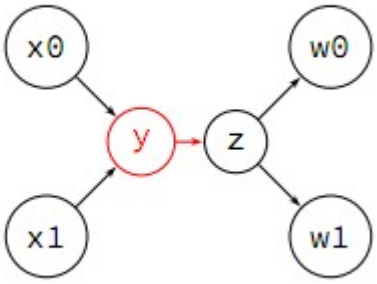
接下来，我们将导数传播到两个输出所依赖的变量 z。我们假设我们已经将反向模式应用于函数 h，并且具有 dh（z，dwdw0，dwdw1）。

```
y = f(x0, x1)
z = g(y)
w0, w1 = h(z)

dwdw0 = {1, 0}
dwdw1 = {0, 1}
dwdz = dh(z, dwdw0, dwdw1)
```



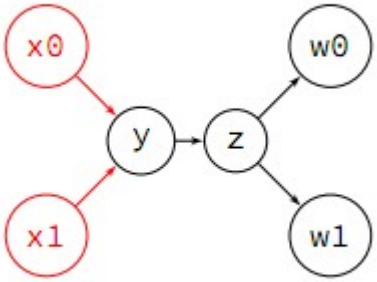
类似地，我们从 z 传播到 y。

```
y = f(x0, x1)
z = g(y)
w0, w1 = h(z)

dwdw0 = {1, 0}
dwdw1 = {0, 1}
dwdz = dh(z, dwdw0, dwdw1)
dwdy = dg(y, dwdz)
```



最后，我们获得了输出 w 相对于两个输入的导数。

```
y = f(x0, x1)
z = g(y)
w0, w1 = h(z)

dwdw0 = {1, 0}
dwdw1 = {0, 1}
dwdz = dh(z, dwdw0, dwdw1)
dwdy = dg(y, dwdz)
dwx0, dwx1 = df(x0, x1, dwdy)
```



反向模式和正向模式之间的一个主要区别使反向模式的实现更加复杂，那就是我们只能在计算出最终输出之后才能开始微分。

自动微分系统的实现

在本节中，我们将讨论自动微分的实际实现。通常，自动微分系统的实现可以被分类为频谱中的一个点，这取决于编译时完成的工作量。在频谱的一端，跟踪方法，有时称为录音方法，每当我们评估函数时，都会重新编译导数。另一方面，源代码转换方法通过只编译一次派生代码，在编译时尽可能多地执行操作。跟踪方法的优点是实现更简单，更容易合并到现有代码中，而源代码转换方法的性能更好，但通常只能处理通用语言的一个子集，实现起来更困难。

在正向模式下处理控制流和递归是很简单的。我们根本不需要修改控制流。然而，在反向模式中，控制流和递归会带来挑战，因为我们需要恢复控制流。考

虑迭代指数示例。要应用反向模式，我们需要恢复 for 循环。我们在这里观察到一个问题：我们需要导数的中间 exp（结果）值。要解决这个问题，我们需要在循环的第一次传递过程中记录中间值：

```
function d_f(x):
    result = x
    results = []
    for i = 1 to 8:
        results.push(result)
        result = exp(result)

    d_result = 1
    for i = 8 to 1:
        // one-based indexing
        d_result = d_result * exp(results[i])
    return d_result
```

在反向模式中转换循环的一般策略是将中间变量推入每个循环的堆栈中，然后在反向循环期间弹出项目。嵌套循环可以以相同的方式处理。为了有效地生成代码，需要进行相关性分析，以便只将稍后使用的变量推送到堆栈。

在堆栈中存储中间变量的相同策略也适用于循环延续、早期退出和条件 while 循环。我们可以使用堆栈的大小作为终止条件。例如，我们将前面的示例修改为 while 循环，并以红色突出显示微分代码：

```
function d_f(x):
    result = x
    results = []
    while result > 0.1 and result < 10:
        results.push(result)
        result = exp(result)

    d_result = 1
    for i = len(results) to 1:
        d_result = d_result * exp(results[i])
    return d_result
```

2.2 从宏观视角理解可微渲染

目前，为了提高渲染的真实度，常常采用基于物理的光照模型，这些物理光照模型让渲染脱离了经验模型的限制，使得渲染出的效果更加逼真。著名的基于物理的光照模型有 Lambert 模型、Phong 模型[1]、Blinn-Phong 反射模型、双向反射分布函数模型（Bidirectional Reflectance Distribution Function，BRDF）等，其中

BRDF 模型可以描述表面指定方向入射光和反射光关系。

渲染在动画、游戏、电影和虚拟现实中有丰富的应用. 传统的渲染理论和方法已经趋于成熟，在工业上已经有许多成熟的软件，比如 OpenGL, 3ds Max 等。在游戏行业，随着图形处理器 (graphics processing unit, GPU) 的发展，已经能够实时渲染高质量高真实感的画面。在动画和电影行业，基于物理的高真实感渲染能够达到以假乱真的效果。在计算机图形学中，一般用三角网格表示三维模型，所以一般的渲染方法都是针对三角网格模型提出的. 除了一般的渲染方法之外，还有许多专门针对某些特殊应用的特殊渲染方法，比如对体素、点云、 毛发[1]和流体等的渲染。

传统渲染管线被抽象为几个固定的步骤，每个步骤有着固定且单一的任务。我们以 OpenGL 使用的可编程渲染管线 (本文中简称为渲染管线) 为例介绍传统渲染管线，它使用的是局部光照模型。如图 2 所示, OpenGL 的渲染管线依次分为顶点着色器、图元装配、几何着色器、光栅化、片段着色器、 测试与混合 6 个步骤. 在三角网格模型渲染过程中，几何着色器仅输出三角片元，假设不开启深度测试和混合，一个简单的渲染管线如图 3 所示，它主要由顶点着色器、光栅化和片段着色器 3 个部分组成。

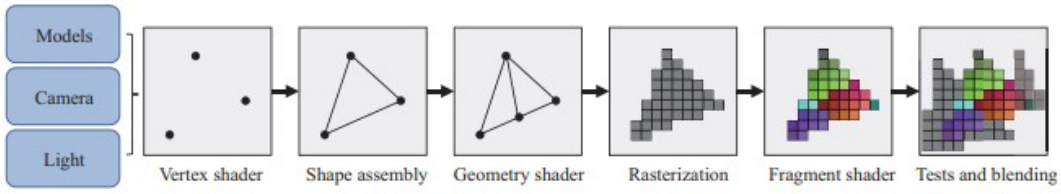


图 1-3 OpenGL 渲染管线

总的来说，可微渲染是可以微分求导的渲染过程，分为正向和逆向的过程，正向过程和传统渲染相同，输入模型和参数得到一张图片，逆向是像素对场景参数求导数，可微渲染需要兼具这两个过程，不仅需要得到渲染结果，还要得到渲染结果对输入的导数。可微渲染不能离开传统的渲染模型，但传统的渲染方法不可微，所以可微渲染往往是基于某种传统渲染模型，通过引入新的技术，使得我们可以得到渲染结果对输入的导数。主流的可微渲染方法往往基于以下两类思想，一类是使用近似的方法，求得近似导数用于反向传播；另一类是改编传统渲染模

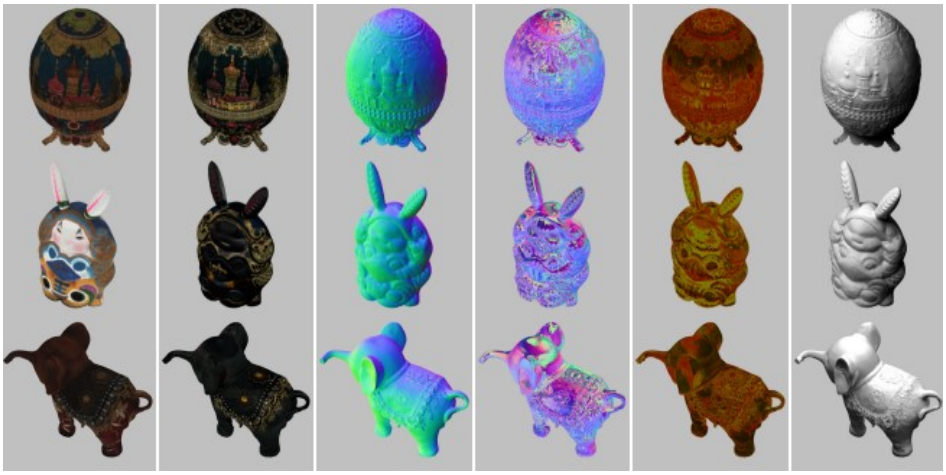
型，让像素对顶点可导。

许多工作使用近似的方法，求得近似导数用于反向传播[1, 2]，或是改编了传统渲染管线中的步骤（一般是光栅化和测试与混合步骤），让像素对顶点可导[3]。求得近似导数用于反向传播的方法没有改变传统渲染的正向过程，这类方法的关键在于如何使导数成为一种有效的、较为真实的近似使得导数对优化输入具有指导意义。改编传统渲染管线的方法通常改编的是传统渲染管线中光栅化这一步骤，因为这一步骤是从连续空间映射到离散空间的步骤，是导致传统渲染不可微分的主要原因。

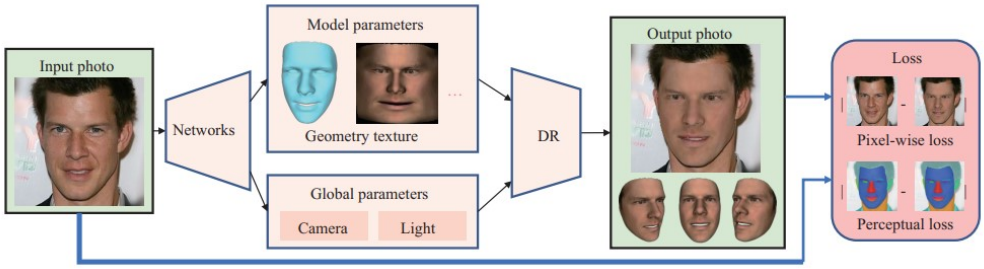
3 可微渲染应用

可微渲染有这广阔的应用前景，它可以通过图片和初值求解出渲染参数，这使得它代替大量建模工作，使自动生成的数据和美术工作者手绘的数据一样易于修改。

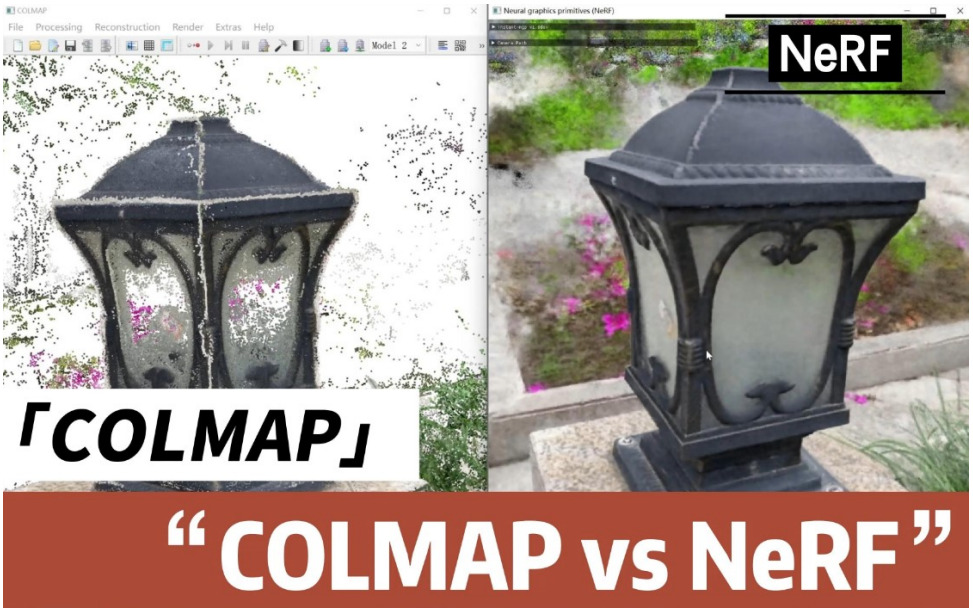
从重建应用场景上来表述，它可以精确重建小型的物体[4]，可以重建人脸这样稍大的物体[6][7][8]，也可以进行景物重建[9][10][11]。



小型物体精确重建[4]



人脸重建[6][7][8]



景物重建[9][10][11]

从重建参数上来表述，它可以重建物体几何及位置参数[12][13][14][15]，可以重建纹理和材质[16][17][18]，可以重建光照参数等。

4 可微渲染展望

局部最小值、过参数化和预处理

当处理噪声和颠簸的函数景观时，已知基于导数的方法是不稳定的。深度学习似乎不支持这一点，很可能是由于它们的参数过多[21,22]。弄清楚如何过度框架化传统算法是真正融合这两个领域的关键。信号处理中的预滤波也可以发挥重要作用：如果我们能够在对函数进行采样之前对其进行平滑处理，这将使导数表现得更好。Yang 和 Barnes[23]暗示了如何为计算机图形中的着色程序自动实现这一点，但将他们的方法推广到任意高维函数需要未来的研究。

在近期的研究中，有诸多学者从其他方面试图解决过参数化的问题。

在文献[4]中，学者们试图通过对空间中同一物体的不同光源不同视角下的采样结果来重建参数已解决参数过多导致的二义性问题。

在文献[20]中，学者们试图通过偏振片获取偏振照明条件下有镜面反射光和无镜面反射光的采样结果，从而将不同参数的重建过程分离到不同阶段中，以解决同时重建多个参数时参数过多、约束不足的问题。

完全可微分的计算机图形学

虽然我们已经成功地研究了基于物理的渲染和一个小的 uid 模拟示例，但这些只是计算机图形学整个领域的子集。希望使整个 3D 建模、仿真和渲染管道可重复。如本文所示，拥有完全可替代的管道可以实现数据驱动的训练和反向推理。随着计算机图形以极其详细和原则性的模拟来模拟世界，它将在现实世界的应用中发挥巨大的作用。

用于其他计算的自动微分处理器

虽然我们可以自动生成用于图像处理和深度学习运算符的有效梯度代码，但与完全通用的编译器相比，编程模型仍然相对有限_是在编写简洁代码的同时实现高性能所必需的。然而，重要的是要涵盖其他用于自动定义的计算，如树遍历、稀疏或图形数据结构、排序等。

5 小结

依靠传统方法通过美术工作者的经验生成包括模型、材质等在内大量的数据资产显然费时费力，对构建宏大虚拟世界构成了挑战。而近年来的可微渲染技术为我们提供了一个可靠的、能够依靠数据驱动的方法生成大量数据资产的工具，这个工具让游戏世界的构建变得更加方便。

本文介绍了可微渲染的出现背景，宏观原理以及可微渲染的广阔应用。其可以通过数据驱动的特性使得它可以通过图片和初值求解出渲染参数，使自动生成的数据和美术工作者手绘的数据一样易于修改。

本文介绍了可微渲染的难点、研究热点和未来发展方向。其中局部最小值、过参数化和预处理相关的问题尤其值得关注。

参考文献

[1] Genova K, Cole F, Maschinot A, et al. Unsupervised training for 3D morphable model regression. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018

[2] Kato H, Ushiku Y, Harada T. Neural 3D mesh renderer. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018

[3] Liu S, Li T, Chen W, et al. Soft rasterizer: a differentiable renderer for image-based 3D reasoning. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2019

[4] Kang K , Xie C , He C , et al. Learning efficient illumination multiplexing for joint capture of reflectance and shape[J]. ACM Transactions on Graphics (TOG), 2019.

[5] Gotardo, Pfu , et al. "Practical dynamic facial appearance modeling and acquisition." ACM Transactions on Graphics (TOG) (2018).

[6] Genova K, Cole F, Maschinot A, et al. Unsupervised training for 3D morphable model regression. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018

[7] Yu C, Wang J, Peng C, et al. BiseNet: bilateral segmentation network for real-time semantic segmentation.In: Proceedings of the 15th European Conference on Computer Vision, Munich, 2018. 334–349

[8] Neural 3D Video Synthesis From Multi-View Video. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2022

[9] Neural Rays for Occlusion-Aware Image-Based Rendering.In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2022

[10] GeoNeRF: Generalizing NeRF With Geometry Priors.In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2022

[11] SNeRF: Stylized Neural Implicit Representations for 3D Scenes.In: Proceedings of the IEEE Conference on Special Interest Group for Computer GRAPHICS (SIGGRAPH), 2022

[12] Yao S, Hsu T M H, Zhu J Y, et al. 3D-aware scene manipulation via inverse graphics. 2018. ArXiv:1808.09351

[13] Kanazawa A, Tulsiani S, Efros A A, et al. Learning category-specific mesh reconstruction from image collections. 2018. ArXiv:1803.07549

[14] Kato H, Harada T. Learning view priors for single-view 3D reconstruction. 2019. ArXiv:1811.10719

[15] Zuffi S, Kanazawa A, Berger-Wolf T Y, et al. Three-D safari: learning to estimate zebra pose, shape, and texture from images “in the wild”. In: Proceedings of 2019 IEEE/CVF International Conference on Computer Vision, Seoul, 2019. 5358–5367

[16] Gecer B, Ploumpis S, Kotsia I, et al. GANFIT: generative adversarial network fitting for high fi delity 3D face reconstruction. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, 2019. 1155 – 1164

[17] Lee G H, Lee S W. Uncertainty-aware mesh decoder for high fidelity 3D face reconstruction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020

[18] Li X, Liu S, Kim K, et al. Self-supervised single-view 3D reconstruction via semantic consistency. In: Proceedings of European Conference on Computer Vision, 2020

[19] Andreas Griewank and Andrea Walther. Evaluating Derivatives: Principles and Techniques of Algorithmic Diòerentiation. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2008.

[20] Riviere J , Gotardo P , Bradley D , et al. Single-shot high-quality facial geometry and skin appearance capture[J]. ACM Transactions on Graphics, 2020, 39(4).

[21] Jonathan Frankle and Michael Carbin. _e lottery ticket hypothesis: Finding sparse, trainable neural networks. In International Conference on Learning Representations, 2019.

[22] Kenji Kawaguchi and Leslie Pack Kaelbling. Elimination of all bad local minima in deep learning. arXiv:1901.00279, 2019.

[23] Y. Yang and C. Barnes. Approximate program smoothing using mean-variance statistics, with application to procedural shader bandlimiting. Comput. Graph. Forum (Proc. Eurographics), 37(2):443–454, 2018.