3.算法

我们的扭曲算法包含两个步骤：通过缝隙雕刻（Seam Carving）进行局部扭曲步骤，以及基于网格的全局扭曲步骤。局部步骤提供了初步的矩形图像，其主要目的是在输入图像上放置网格（见图2(b)(c)）。全局步骤则优化这个网格，以保留感知属性，包括形状和直线（见图2(d-f)）。

3.1 无网格的局部扭曲

我们的局部扭曲步骤是对缝隙雕刻算法 [Avidan and Shamir 2007] 的修改。原始的缝隙雕刻算法在图像中插入水平或垂直缝隙，从而在垂直或水平方向上扩展图像一个像素。如果允许缝隙不穿过整个图像，我们可以改变图像的边界形状，使其符合矩形边界。最近，已有研究 [Qi and Ho 2012] 提出了使用缝隙雕刻将矩形图像裁剪成不规则形状（如圆形或椭圆形）。这与我们的任务正好相反。

假设不规则输入图像的边界框是目标矩形边界。我们定义"边界段"是目标矩形边界四个边（上/下/左/右）中缺失像素的连续序列。图3(i) 显示了一个例子。每次我们选择最长的边界段并插入一个缝隙。例如，当边界段位于右侧（如图3(i)），我们在图像中插入一个垂直的非穿透缝隙。这个缝隙与所选边界段有相同的起始和结束y坐标（见图3(ii)）。然后我们将这个缝隙右侧的所有像素向右移动一个像素。上/下/左侧缝隙的情况可以类似处理。我们以这种方式反复插入缝隙（见图3(iii)(iv)），直到矩形边界上没有缺失像素。

为了找到非穿透缝隙，我们在"子图像"中运行缝隙雕刻算法。例如，图3(ii) 中的子图像与边界段有相同的起始和结束y坐标。然后我们在这个子图像上应用缝隙雕刻算法（在我们的实现中采用了改进的缝隙雕刻 [Rubinstein et al. 2008]）。由于子图像可能包含缺失像素，我们为这些像素分配一个无限成本 (10^8) 来防止缝隙通过它们。

从填补未知区域的角度来看，插入一个缝隙将减少图像中缺失像素的数量（这个数量等于缝隙上的像素数）。从扭曲的角度来看，插入一个缝隙相当于计算一个位移场u(x)。我们用x = (x, y) 表示输出像素的坐标，用u = (ux, uy) 表示位移。输出像素值可以通过扭曲输入图像得到：$\mathrm{out}(x) = I_{\mathrm{in}}(x + u(x))$其中，$I_{in}$ 和 I_{out}分别代表输入和当前输出图像。对于图3(ii) 中的例子，位移u 对于这个缝隙右侧的所有像素为 (-1, 0)，对其他所有像素为零。随着所有缝隙的计算，我们相应地获得一个位移场u。我们将这个步骤称为局部扭曲，因为扭曲主要分布在缝隙附近（见图4）。

3.2 基于网格的全局变形

为了生成一个矩形图像，同时保持直线和形状等高级属性，我们基于网格优化一个全局能量。

3.2.1 网格放置

为了在输入的不规则图像上生成网格，我们首先在局部变形步骤的矩形结果上放置一个网格（图2(c)）。本文中我们使用了一个大约有400个顶点的网格。我们使用局部变形的位移场将所有网格顶点反向变形回输入图像域。这样我们就得到了一个放置在输入图像上的网格（图2(d)）。全局变形步骤基于这个网格。局部变形的结果图像随后被丢弃。

3.2.2 能量函数

我们设计了一个能量函数，该函数在保持形状和直线的同时施加矩形边界约束。我们的能量定义部分基于 [Chang and Chuang 2012]，但形式更简单，涉及的参数更少。
我们将网格 V 参数化为 $\{v_i\}$，其中 $v_i = (x_i, y_i)$ 是网格顶点的位置。我们将输入网格表示为 $\hat{V}$。我们优化关于输出网格 V 的能量函数。

形状保持

我们的形状保持能量 E_S 鼓励每个四边形进行相似变换（即平移+旋转+缩放），如[Zhang et al. 2009; Chang and Chuang 2012]所用。其定义为：

$$E_S(V) = \frac{1}{N}\sum_q \|(A_q(A_q^TA_q)^{-1}A_q^T - I)V_q\|^2. \quad (2)$$

这里 N 是网格中的四边形数量，q 是四边形索引，I 是单位矩阵，A_q 是一个 8×4 的矩阵，而 V_q 是四边形上的一个 8×1 的向量：

$$A_q = \begin{bmatrix} \hat{x}_0 & -\hat{y}_0 & 1 & 0 \\ \hat{y}_0 & \hat{x}_0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \hat{x}_3 & -\hat{y}_3 & 1 & 0 \\ \hat{y}_3 & \hat{x}_3 & 0 & 1 \end{bmatrix}, \quad V_q = \begin{bmatrix} x_0 \\ y_0 \\ \vdots \\ x_3 \\ y_3 \end{bmatrix}. \quad (3)$$

这里我们使用 $(x_0, y_0), \ldots, (x_3, y_3)$ 表示输出四边形的四对坐标，用 $(\hat{x}_0, \hat{y}_0), \ldots, (\hat{x}_3, \hat{y}_3)$ 表示输入四边形的四对坐标。E_S 的推导可以在 [Zhang et al. 2009] 中找到。E_S 是 V 的二次函数。
与之前的工作不同，我们没有在形状保持项中引入显著性权重。这是因为全景图像通常覆盖多种内容，并且不包含特别显著的对象。

线条保持
我们按照 [Chang and Chuang 2012] 的方式来保持线条。我们的线条保持能量 E_L 鼓励直线保持直线，平行线保持平行。我们使用 [Von Gioi et al. 2010] 的代码检测输入图像中的线段。我们将所有检测到的线段与输入网格的边缘相交，因此每个生成的线段都位于一个四边形内。我们将线段方向范围 $[-\pi/2, \pi/2)$ 量化为 $M = 50$ 个区间。为了保持直线和平行，我们鼓励同一区间内的所有线段共享一个共同的旋转角度 θ。线条保持项 E_L 包含所有这些角度 $\{\theta_m\}_{m=1}^M$。
给定一个线段，我们使用其两个端点的差向量计算其方向向量（带比例）。如果我们将一个线段的两个端点表示为其四边形顶点 V_q 的双线性插值，我们可以将方向向量表示为 V_q 的线性函数。我们将这个线段的输入方向向量表示为 $\hat{e}$。给定目标旋转角度 θ_m，我们希望最小化线段的以下变形：

$$\|sR\hat{e} - e\|^2, \quad (4)$$

其中 R 是旋转矩阵，s 是线段的缩放因子。关于 s 的最小化得到：$s = (\hat{e}^T\hat{e})^{-1}\hat{e}^TR^Te$。将 s 代入上述公式，我们可以看出变形是 e 的二次函数：

$$\|Ce\|^2, \quad (5)$$

其中矩阵 C 是

$$C = R\hat{e}(\hat{e}^T\hat{e})^{-1}\hat{e}^TR^T - I. \quad (6)$$

因为 e 是 V_q 的线性函数，变形可以写成 V_q 的二次函数。
我们的线条保持能量 E_L 定义为所有线段的平均变形：

$$E_L(V, \theta_m) = \frac{1}{N_L}\sum_j \|C_j(\theta_m(j))e_q(j)\|^2, \quad (7)$$

其中 N_L 是线段的数量。线段用 j 索引，$q(j)$ 是包含它的四边形。矩阵 $C_j(\theta_m(j))$ 依赖于包含该线段的区间的期望旋转角度 $\theta_m(j)$。E_L 是 V 的二次函数。
与 [Chang and Chuang 2012] 不同，我们的能量与每个线段的缩放因子 (s) 和平移解耦。因此我们的能量有更少的变量和参数。

边界约束
我们希望将网格边界上的顶点拖动到一个矩形上。边界约束项 E_B 简单定义为：

$$E_B(V) = \sum_{v_i \in L} x_i^2 + \sum_{v_i \in R} (x_i - w)^2 + \sum_{v_i \in T} y_i^2 + \sum_{v_i \in B} (y_i - h)^2. \quad (8)$$

这里 $L/R/T/B$ 分别表示左/右/上/下边界顶点，w/h 表示目标矩形（例如边界框）的宽度/高度。注意我们只约束每个边界顶点的两个坐标中的一个，例如顶部边界的顶点可以水平移动。

总能量
我们的总能量函数 E 是：

$$E(V, \theta_m) = E_S(V) + \lambda_L E_L(V, \theta_m) + \lambda_B E_B(V), \quad (9)$$

其中 λ_L 和 λ_B 是两个权重。我们将边界权重 λ_B 设为无穷大 (10^8) 以施加硬边界约束。
线条保持权重 λ_L 是我们算法中的主要影响参数。在实验中，我们发现当 λ_L 足够大（例如 ≥ 10）时效果稳定良好。这意味着线条保持 (E_L) 的重要性高于形状保持 (E_S)。这表明人眼对弯曲的直线比对变形的形状更敏感。在本文中我们固定 $\lambda_L = 100$。图 5 显示了线条保持的效果。

3.2.3高效优化

我们采用交替算法来最小化 $(E(V, \{\theta_m\})$。初始化由局部变形结果给出（即简单的规则网格）。我们运行以下交替算法 10 次：

1. 固定 θ_m 并更新 (V)。在这种情况下，E 是 V二次函数，可以通过解线性系统来优化。因为 (V) 中只有几百个顶点，这一步的运行时间可以忽略不计（在 C++ 实现中）。
2. 固定 (V) 并更新 θ_m 。因为θ_m 是彼此独立的，我们可以分别优化每个θ_m 。在这种情况下，我们最小化：

$$\min_{\theta_m} \sum_{j \in \text{bin}(m)} \|C_j(\theta_m) e_q(j)\|^2 \quad (10)$$

这个问题可以通过牛顿法等迭代求解器来优化。相反，我们采用一种由此能量函数直觉驱动的简单非迭代解法。公式 (10) 的直观意义是为第 (m) 个区间中的所有线段找到一个共同的旋转角度θ_m ，使得 θ_m 近似于任意线段 e_j与其对应的 $\hat{e_j}$ 之间的相对角度。因此我们简单地计算第 (m) 个区间中所有线段的 e_j与 $\hat{e_j}$ 的相对角度，并将它们的平均值作为 θ_m 。我们发现这对最小化 (10) 是一个足够好的解。

3.2.4拉伸减少与后处理

我们假设边界框是目标矩形边界。但如果这个矩形的长宽比不合适，输出图像可能会显得拉伸。这个问题在透视投影的情况下尤为明显（见图 6）。为了减少拉伸，我们在全局变形步骤后更新目标矩形。对于每个网格四边形，我们通过以下公式计算其 (x) 缩放因子 (s_x)：

$$s_x = \frac{\hat{x}_{\max} - \hat{x}_{\min}}{x_{\max} - x_{\min}}$$

(x) 缩放因子的平均值 $\bar{s_x}$ 是所有四边形 s_x的平均值。然后目标矩形的宽度按 $1/\bar{s_x}$ 缩放。高度同样使用 y 缩放因子的平均值 $\bar{s_y}$ 进行处理。我们使用这个更新后的目标矩形再次运行全局变形步骤。图 6 显示了这一步减少了拉伸变形。

使用优化后的网格，我们通过双线性插值从四边形顶点的位移计算任何像素的位移。但这可能偶尔会在边界上留下少量缺失的像素，因为网格线可能无法很好地匹配不规则的输入边界。我们简单地使用最近的已知像素的颜色填充每个缺失的像素。

3.3实现与速度

由于用于变形的位移图大多是平滑的，我们在较小的图像尺度上计算它。我们首先将输入图像下采样到固定大小（1 兆像素）。局部/全局变形步骤在下采样的图像上运行。然后我们通过双线性插值对结果的矩形位移图进行上采样。我们使用这个位移图来变形全分辨率的输入图像。

在我们的实现中（C++，单核），算法在一台配备 Intel Core i7 2.9GHz CPU 和 8GB 内存的 PC 上处理一张 10 兆像素的全景图像（图 1）需要 1.5 秒。运行时间主要集中在使用计算的位移图变形全分辨率图像上。相比之下，Adobe Photoshop 中的图像补全工具“内容感知填充”处理这张缺失 18% 像素的 10 兆像素图像需要 19.1 秒。

4.结果

我们在各种真实案例中展示了我们的结果，见图1、2、7、8和9。我们的方法成功地保持了内容而没有引入明显的伪影。通常，图7（上）和图9（上）是两个具有挑战性但实际的360°全景图像。我们认为我们的方法在创建此类引人注目的照片方面特别有优势。

在图1和7中，我们与Adobe Photoshop CS5的“内容感知填充”进行了比较。这种图像完成工具基于 [Wexler et al. 2007; Barnes et al. 2009]（据 [Adobe 2009] 介绍）。在图8中，我们与另一种最近的完成技术 [Kopf et al. 2012] 在作者提供的公开图像/结果上进行了比较。我们发现这些最先进的图像完成技术在合成语义合理的内容方面存在局限性，例如图1(b) 和图8(左) 中的建筑物。它们也可能将语义内容视为纹理，例如图7(底部) 的城市景观。我们还在图1和7中展示了裁剪结果。我们看到裁剪可能会严重限制视野。

图10显示了结合图像完成和裁剪的示例。在这种情况下，我们手动裁剪图像完成结果，使用一个窗口来去除明显的伪影。但我们看到这个操作仍然会去除大量的图像内容。

用户研究。为了收集大量具有不规则边界的全景图像进行用户研究，我们从MIT SUN360数据库 [Xiao et al. 2012] 构建了一个“半合成”数据集。该数据库包含由互联网用户贡献的10,405个真实的全景（360°×180°）场景。这些场景被标注为80个类别，涉及室内/室外案例和各种人工/自然场景（例如卧室、街道、山脉等）。如果某个类别包含超过5个场景，我们使用前5个场景（按照数据库中的原始顺序），并获得一个包含367个场景的子集。

Xiao et al. [2012] 提供了模拟用普通相机在全景场景中拍照的代码2。因此，我们可以模拟捕捉照片序列以进行拼接的行为。我们使用三种方式来合成这些序列（见图11上部）。我们生成用于圆柱投影的5×1图像数组，用于透视投影的3×1图像数组，以及用于球面投影的3×2图像数组。相邻图像随机有30-50%的重叠区域。每张图像的相机位置略有随机扰动，以模拟随意的相机移动。我们为每个全景场景合成3个序列（使用指定的投影拼接）。因此，我们的数据集包含1,101个（367×3）拼接图像。一些示例见图11。

我们在这个数据集中将我们的扭曲策略与图像完成策略进行了比较。我们选择“内容感知填充”作为图像完成工具。每次在一个屏幕上显示三张图像：输入图像、我们的扭曲结果和完成结果。用户可以放大图像。用户需要回答是更喜欢我们的结果还是完成结果，或者没有偏好。在“没有偏好”的情况下，用户需要回答两个结果是“都好”还是“都不好”。我们的研究有10名参与者，包括5名具有计算机图形/视觉背景的研究人员/学生和5名来自其他领域的志愿者。

图12显示了用户研究结果。我们的方法被显著地更喜欢。这表明了扭曲策略在矩形化全景图方面的优势。我们还看到在透视投影中我们的优势更为显著。这是因为在透视情况下，缺失区域通常较大，图像完成的成功率较低。

5.限制和未来工作

当不规则边界是投影和随意相机移动的结果时，我们的方法效果很好。但我们注意到我们的方法在以下方面存在局限性。

(i) 当场景未完全拍摄时，例如在图像数组中缺少几张图像时，我们的方法可能会失败。在这种情况下，我们的方法可能导致明显的失真（见图13）。

(ii) 我们的方法可能会扭曲非常凹的边界附近的内容（见图14(a)(b)）。为了减少凹度，可以手动引入一个透明区域，将图像扭曲时将该区域视为已知像素，扭曲后使用图像完成填充该区域（见图14(c)）。

(iii) 我们的方法可能会弯曲未检测到的线条。这可以通过一些用户交互来解决。

(iv) 当局部区域有大量线条时，我们的方法可能无法保留所有线条。这对于扭曲方法来说是一个常见的挑战。

我们的方法是纯粹基于内容的，因此在拼接步骤中的知识不可用时可以广泛应用。但如果允许在拼接步骤中考虑矩形化问题，这将是一个有趣的问题。我们将这个问题留待未来研究。