

# Image Processing:

## View and Image Morphing

CS334

### Motivation - Rendering from Images



- Given
  - left image
  - right image
- Create intermediate images
  - simulates camera movement

[Seitz96]

### Related Work

- Panoramas ([Chen95/QuicktimeVR], etc)
  - user can look in any direction at few given locations but camera translations are *not* allowed...

Quicktime VR Demo

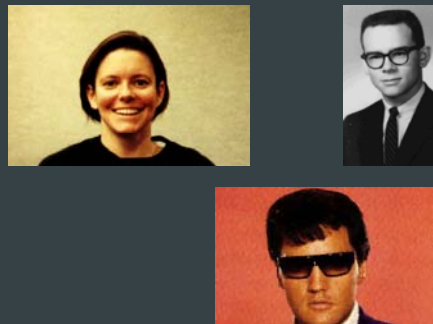
### Overview

- Image morphing
- View morphing
  - image pre-warping
  - image morphing
  - image post-warping

### Overview

- Image morphing
- View morphing
  - image pre-warping
  - image morphing
  - image post-warping

### Image Morphing Examples



## Image Morphing

- Identify correspondences between input/output image
- Produce a sequence of images that allow a smooth transition from the input image to the output image

## Image Morphing

1. Correspondences



## Image Morphing

1. Correspondences



## Image Morphing

1. Correspondences



## Image Morphing

1. Correspondences

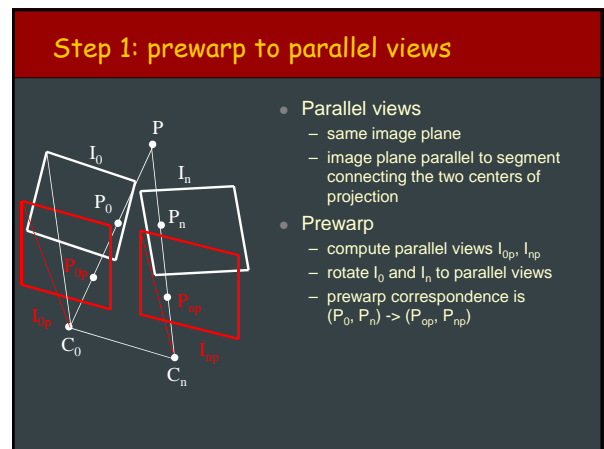
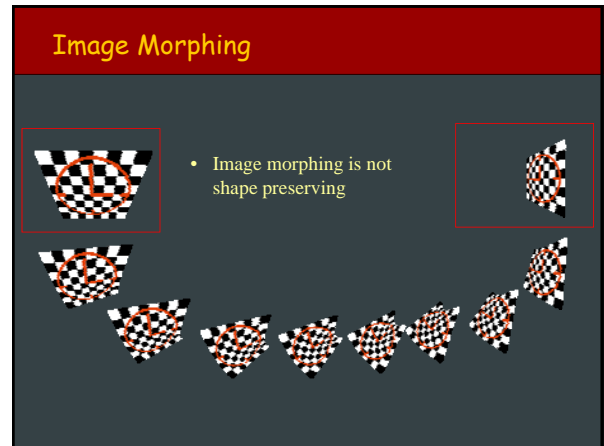
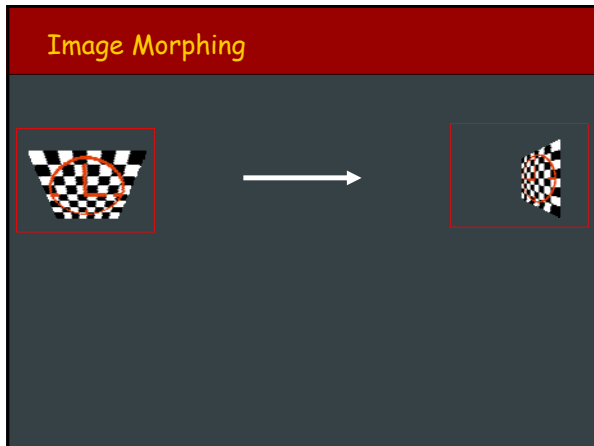


## Image Morphing

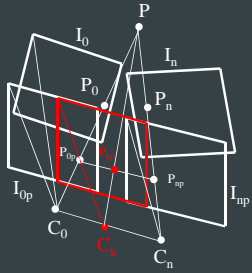
1. Correspondences
2. Linear interpolation



$$\dot{P}_k = \left(1 - \frac{k}{n}\right) \dot{P}_0 + \frac{k}{n} \dot{P}_n$$

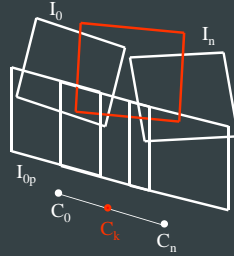


## Step 2: morph parallel images



- Shape preserving
- Use prewarped correspondences
- Interpolate  $C_k$  from  $C_0$   $C_n$

## Step 3: Postwarping

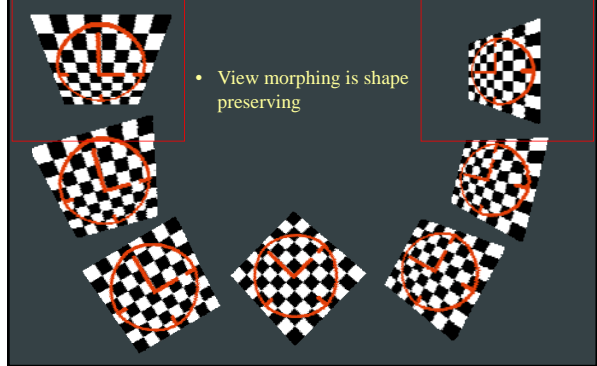


- Postwarp morphed image
  - create intermediate view
    - $C_k$  is known
    - interpolate view direction and tilt
  - rotate morphed image to intermediate view

## View morphing



## View morphing



## View Morphing More Examples

- Using computer vision/stereo reconstruction techniques

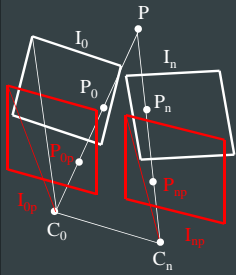


## Overview

- Image morphing
- View morphing, more details for synthetic rendering
  - image pre-warping
  - image morphing
  - image post-warping

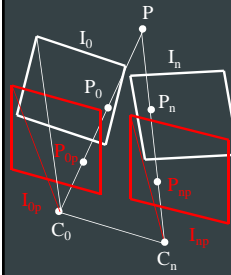
### Step 1: prewarp to parallel views

- Parallel views



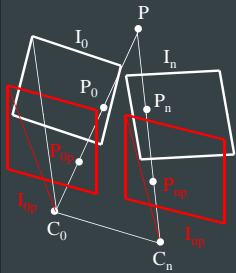
### Step 1: prewarp to parallel views

- Parallel views
  - use  $C_0 C_n$  for  $x$  ( $a_p$  vector)



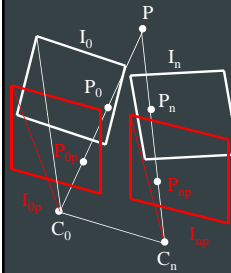
### Step 1: prewarp to parallel views

- Parallel views
  - use  $C_0 C_n$  for  $x$  ( $a_p$  vector)
  - use  $(a_0 \times b_0) \times (a_n \times b_n)$  as  $y$  ( $-b_p$ )



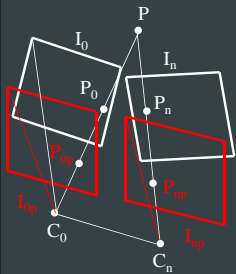
### Step 1: prewarp to parallel views

- Parallel views
  - use  $C_0 C_n$  for  $x$  ( $a_p$  vector)
  - use  $(a_0 \times b_0) \times (a_n \times b_n)$  as  $y$  ( $-b_p$ )
  - use  $z = x$  cross  $y$



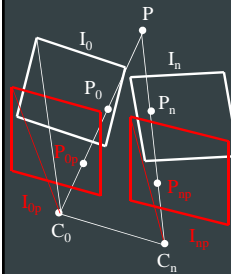
### Step 1: prewarp to parallel views

- Parallel views
  - use  $C_0 C_n$  for  $x$  ( $a_p$  vector)
  - use  $(a_0 \times b_0) \times (a_n \times b_n)$  as  $y$  ( $-b_p$ )
  - use  $z = x$  cross  $y$
  - use same pixel size
  - use wider field of view

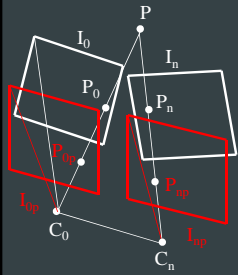


### Step 1: prewarp to parallel views

- prewarping using reprojection of rays

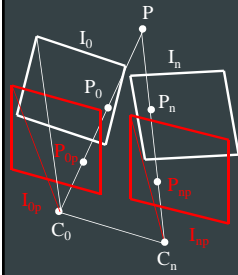


### Step 1: prewarp to parallel views



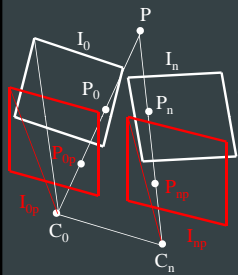
- prewarping using reprojection of rays
  - look up all the rays of the prewarped view in the original view

### Step 1: prewarp to parallel views



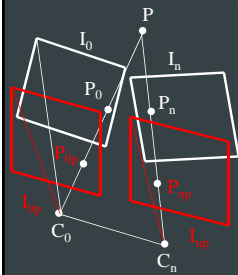
- prewarping using reprojection of rays
  - look up all the rays of the prewarped view in the original view
- alternative: prewarping using texture mapping

### Step 1: prewarp to parallel views



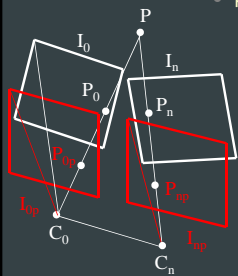
- prewarping using reprojection of rays
  - look up all the rays of the prewarped view in the original view
- alternative: prewarping using texture mapping
  - create polygon for image plane
  - consider it texture mapped with the image itself
  - render "scene" from prewarped view
  - if you go this path you will have to implement clipping with prewarped plane
  - note: texture mapping in hardware

### Step 1: prewarp to parallel views



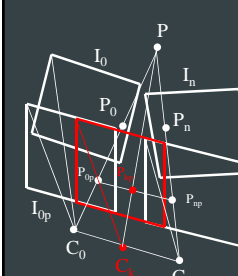
- prewarping correspondences
  - for all pairs of correspondence  $P_0 P_n$

### Step 1: prewarp to parallel views



- prewarping correspondences
  - for all pairs of correspondence  $P_0 P_n$ 
    - project  $P_0$  on  $I_{op}$ , computing  $P_{0p}$
    - project  $P_n$  on  $I_{op}$ , computing  $P_{np}$
    - prewarped correspondence is  $P_{0p} P_{np}$

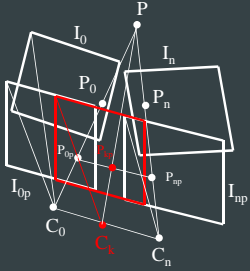
### Step 2: morph parallel images



- Image morphing
  - use prewarped correspondences to compute a correspondence for all pixels in  $I_{op}$
  - linearly interpolate  $I_{op}$  to intermediate positions

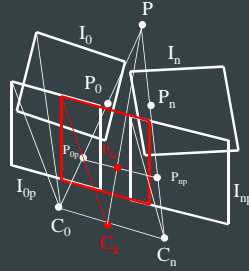
## Step 2: morph parallel images

- Image morphing
  - use prewarped correspondences to compute a correspondence for all pixels in  $I_{op}$
  - linearly interpolate  $I_{op}$  to intermediate positions
  - useful observation
    - corresponding pixels are on same line in prewarped views



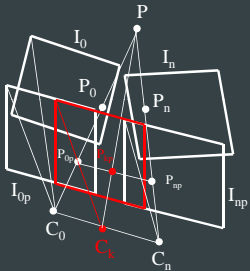
## Step 2: morph parallel images

- Image morphing
  - use prewarped correspondences to compute a correspondence for all pixels in  $I_{op}$
  - linearly interpolate  $I_{op}$  to intermediate positions
  - useful observation
    - corresponding pixels are on same line in prewarped views
  - preventing holes



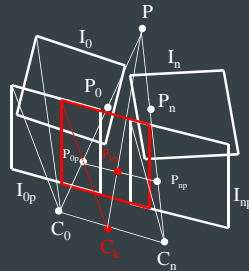
## Step 2: morph parallel images

- Image morphing
  - use prewarped correspondences to compute a correspondence for all pixels in  $I_{op}$
  - linearly interpolate  $I_{op}$  to intermediate positions
  - useful observation
    - corresponding pixels are on same line in prewarped views
  - preventing holes
    - use larger footprint (e.g., 2x2)
    - or linearly interpolate between consecutive samples
    - or postprocess morphed image looking for background pixels and replacing them with neighboring values



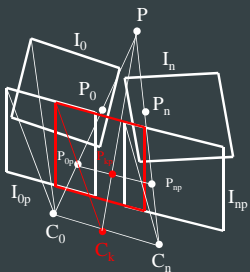
## Step 2: morph parallel images

- Image morphing
  - use prewarped correspondences to compute a correspondence for all pixels in  $I_{op}$
  - linearly interpolate  $I_{op}$  to intermediate positions
  - useful observation
    - corresponding pixels are on same line in prewarped views
  - preventing holes
    - use larger footprint (e.g., 2x2)
    - or linearly interpolate between consecutive samples
    - or postprocess morphed image looking for background pixels and replacing them with neighboring values
  - visibility artifacts
    - collision of samples



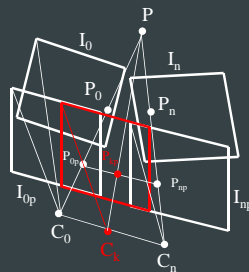
## Step 2: morph parallel images

- Image morphing
  - use prewarped correspondences to compute a correspondence for all pixels in  $I_{op}$
  - linearly interpolate  $I_{op}$  to intermediate positions
  - useful observation
    - corresponding pixels are on same line in prewarped views
  - preventing holes
    - use larger footprint (e.g., 2x2)
    - or linearly interpolate between consecutive samples
    - or postprocess morphed image looking for background pixels and replacing them with neighboring values
  - visibility artifacts
    - collision of samples
      - Zbuffer of disparity

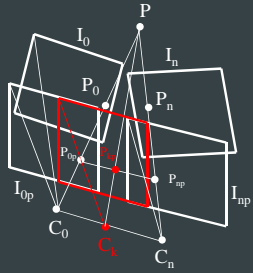


## Step 2: morph parallel images

- Image morphing
  - use prewarped correspondences to compute a correspondence for all pixels in  $I_{op}$
  - linearly interpolate  $I_{op}$  to intermediate positions
  - useful observation
    - corresponding pixels are on same line in prewarped views
  - preventing holes
    - use larger footprint (e.g., 2x2)
    - or linearly interpolate between consecutive samples
    - or postprocess morphed image looking for background pixels and replacing them with neighboring values
  - visibility artifacts
    - collision of samples
      - Zbuffer of disparity
    - holes

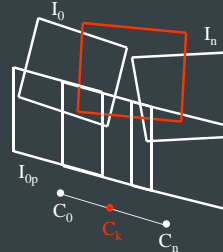


## Step 2: morph parallel images



- Image morphing
  - use prewarped correspondences to compute a correspondence for all pixels in  $I_{0p}$
  - linearly interpolate  $I_{0p}$  to intermediate positions
- useful observation
  - corresponding pixels are on same line in prewarped views
- preventing holes
  - use larger footprint (e.g., 2x2)
  - or linearly interpolate between consecutive samples
  - or postprocess morphed image looking for background pixels and replacing them with neighboring values
- visibility artifacts
  - collision of samples
    - Zbuffer of disparity
  - holes
    - morph  $I_{0p}$  to  $I_{kp}$
    - use additional views

## Step 3: Postwarping



- create intermediate view
  - $C_k$  is known
  - current view direction is a linear interpolation of the start and end view directions
  - current up vector is a linear interpolation of the start and end up vectors
- rotate morphed image to intermediate view
  - same as prewarping