Recap and Outline

- Ready/Valid Handshaking
- async FIFO
- SIFT Algorithm, part 1

Outline for Today

- SIFT Algorithm, conclusion
- FSM implementation
- FSM Moore vs Mealy
- FSM State Assignment
Feature Tracking Project

The SIFT (Scale Invariant Feature Transform) Detector and Descriptor

- developed by David Lowe
- University of British Columbia
- US patent


http://demo.ipol.im/demo/82/wait?key=ECE94E2AEE6F0D1CCD5265DB4E69D224&show=antmy_detect&action=cust_sift_matching

Slides courtesy of Prof. Linda Shapiro, Dept. of CSE, U. Washington
Idea of SIFT

• Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters.

Lowe’s Pyramid Scheme

The parameter $s$ determines the number of images per octave.

Slides courtesy of Prof. Linda Shapiro, Dept. of CSE, U. Washington
Gaussian Smoothing Calculation


review from lec14

Gaussian Smoothing

2. Key point localization

- Detect maxima and minima of difference-of-Gaussian in scale space
- Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below
- Total comparisons?

For each max or min found, output is the location and the scale.

- Lowe, Fig. 2
Overall Procedure at a High Level

1. **Scale-space extrema detection**
   Search over multiple scales and image locations.

2. **Keypoint localization**
   Fit a model to determine location and scale.
   Select keypoints based on a measure of stability.

3. **Orientation assignment**
   Compute best orientation(s) for each keypoint region.

4. **Keypoint description**
   Use local image gradients at selected scale and rotation to describe each keypoint region.

**Scale-space extrema detection: experimental results over 32 images that were synthetically transformed and noise added.**

- **Sampling in scale for efficiency**
  - How many scales should be used per octave? $S = ?$
    - More scales evaluated, more keypoints found
    - $S < 3$, stable keypoints increased too
    - $S > 3$, stable keypoints decreased
    - $S = 3$, maximum stable keypoints found

Slides courtesy of Prof. Linda Shapiro, Dept. of CSE, U. Washington

Lowe, Fig.3
Keypoint localization

- Once a keypoint candidate is found, perform a detailed fit to nearby data to determine
  - location, scale, and ratio of principal curvatures
- In initial work keypoints were found at location and scale of a central sample point.
- In newer work, they fit a 3D quadratic function to improve interpolation accuracy.
- The Hessian matrix was used to eliminate edge responses.

Slides courtesy of Prof. Linda Shapiro, Dept. of CSE, U. Washington

Corners as distinctive interest points

\[ M = X \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} X^T \]

The eigenvalues of M reveal the amount of intensity change in the two principal orthogonal gradient directions in the window.

- **"edge":** \( \lambda_1 >> \lambda_2 \)
- **"corner":** \( \lambda_1 \) and \( \lambda_2 \) are large, \( \lambda_1 \approx \lambda_2 \);
- **"flat" region** \( \lambda_1 \) and \( \lambda_2 \) are small;

One way to score the cornerness:

\[ f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} \]

see Lowe ‘04 paper for details.

slide credit:
CS 143, Brown Univ
James Hays, 2011
3. Orientation assignment

- Create histogram of local gradient directions at selected scale
- Assign canonical orientation at peak of smoothed histogram
- Each key specifies stable 2D coordinates \((x, y, \text{scale,orientation})\)

If 2 major orientations, use both.

Keypoint localization with orientation

Lowe, Fig.5

Slides courtesy of Prof. Linda Shapiro, Dept. of CSE, U. Washington
4. Keypoint Descriptors

- At this point, each keypoint has
  - location
  - scale
  - orientation
- Next is to compute a descriptor for the local image region about each keypoint that is
  - highly distinctive
  - as invariant as possible to variations such as changes in viewpoint and illumination
- Normalization
  - Rotate the window to standard orientation
  - Scale the window size based on the scale at which the point was found.

Lowe’s Keypoint Descriptor
(shown with 2 X 2 descriptors over 8 X 8)

In experiments, 4x4 arrays of 8 bin histogram is used, a total of 128 features for one keypoint

Slides courtesy of Prof. Linda Shapiro, Dept. of CSE, U. Washington
Biological Motivation

• Mimic complex cells in primary visual cortex
• Hubel & Wiesel found that cells are sensitive to orientation of edges, but insensitive to their position
• This justifies spatial pooling of edge responses

[“Eye, Brain and Vision” – Hubel and Wiesel 1988]

Lowe’s Keypoint Descriptor

• use the normalized region about the keypoint
• compute gradient magnitude and orientation at each point in the region
• weight them by a Gaussian window overlaid on the circle
• create an orientation histogram over the 4 X 4 subregions of the window
• 4 X 4 descriptors over 16 X 16 sample array were used in practice. 4 X 4 times 8 directions gives a vector of 128 values.
Automatic mosaicing

http://www.cs.ubc.ca/~mbrown/autostitch/autostitch.html

Uses for SIFT

• Feature points are used also for:
  - Image alignment (homography, fundamental matrix)
  - 3D reconstruction (e.g. Photo Tourism)
  - Motion tracking
  - Object recognition
  - Indexing and database retrieval
  - Robot navigation
  - … many others

[ Photo Tourism: Snavely et al. SIGGRAPH 2006 ]

Slides courtesy of Prof. Linda Shapiro, Dept. of CSE, U. Washington
FSM Implementation

- Flip-flops form state register
- Number of states \(2^n\) for \(n\) flip-flops
- CL (combinational logic) calculates next state and output
- Remember: The FSM follows exactly one edge per cycle.

FSM Formal Design Process

Review of Design Steps:

1. Specify circuit function (English)
   1.1 choose Moore or Mealy
2. Draw state transition diagram
3. Write down symbolic state transition table (case stmt)
4. Assign states
5. Derive logic equations [output decoder, next state decoder]
6. Derive circuit diagram
   - Register to hold state
   - Combinational Logic for Next State and Outputs
State Encoding

• In general:
  # of possible FSM state = 2\(^\#\) of Flip-flops
  Example:
  state1 = 01, state2 = 11, state3 = 10, state4 = 00

• However, often more than \(\log_2(\# \text{ of states})\) FFs are used, to simplify logic at the cost of more FFs.

• Extreme example is one-hot state encoding.

State Encoding

• One-hot encoding of states.
• One FF per state.

Ex: 3 States

STATE1: 001
STATE2: 010
STATE3: 100

FF1 FF2 FF3

• Why one-hot encoding?
  - Simple design procedure.
    • Circuit matches state transition diagram (example next page).
    • Often can lead to simpler and faster “next state” and output logic.

• Why not do this?
  - Can be costly in terms of Flip-flops for FSMs with large number of states.
  - FPGA costs? Hint NSD…

• FPGAs are “Flip-flop rich”, therefore one-hot state machine encoding is often a good approach.
One-hot encoded combination lock

FSM Implementation Notes

• General FSM form:

• All examples so far generate output based only on the present state:

• Commonly named Moore Machine
  (If output functions include both present state and input then called a Mealy Machine)
Finite State Machines

- **Example: Edge Detector**
  
  Bit are received one at a time (one per cycle), such as: 00011010. Design a circuit that asserts its output for one cycle when the input bit stream changes from 0 to 1.

  Try two different solutions.

 State Transition Diagram Solution A

<table>
<thead>
<tr>
<th>IN</th>
<th>PS</th>
<th>NS</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>01</td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>
**Solution B**

*Output depends not only on PS but also on input, IN*

<table>
<thead>
<tr>
<th>ZERO=0, ONE=1</th>
<th>IN</th>
<th>PS</th>
<th>NS</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

NS = IN, OUT = IN \( PS' \)

What's the intuition about this solution?

Notation:
Input condition/output

---

**Edge detector timing diagrams**

- Solution A: output follows the clock
- Solution B: output changes with input rising edge and is asynchronous wrt the clock.
**FSM Comparison**

*Solution A*

**Moore Machine**
- output function only of PS
- maybe more states (why?)
- synchronous outputs
  - no glitches
  - one cycle "delay"
  - full cycle of stable output

*Solution B*

**Mealy Machine**
- output function of both PS & input
- maybe fewer states
- asynchronous outputs
  - if input glitches, so does output
  - output immediately available
  - output may not be stable long enough to be useful (below):

If output of Mealy FSM goes through combinational logic before being registered, the CL might delay the signal and it could be missed by the clock edge.

**FSM Recap**

Both machine types allow one-hot implementations.
Final Notes on Moore versus Mealy

1. A given state machine could have both Moore and Mealy style outputs. Nothing wrong with this, but you need to be aware of the timing differences between the two types.

2. The output timing behavior of the Moore machine can be achieved in a Mealy machine by “registering” the Mealy output values:

![Diagram of Mealy Machine with Output Register]

General FSM Design Process with Verilog Implementation

Design Steps:

1. Specify **circuit function** (English)
2. Draw **state transition diagram**
3. Write down **symbolic state transition table**
4. Assign encodings (bit patterns) to symbolic states
5. Code as Verilog behavioral description

- Use parameters to represent encoded states.
- Use separate always blocks for register assignment and CL logic block.
- Use case for CL block. Within each case section assign all outputs and next state value based on inputs. **Note:** For Moore style machine make outputs dependent only on state not dependent on inputs.
Conclusions

• SIFT details
• State assignment
• Moore Machine
• Mealy Machine
Eliminating the Edge Response

- Reject flats:
  \[ |D(x)| < 0.03 \]
- Reject edges:
  \[
  H = \begin{bmatrix}
  D_{xx} & D_{xy} \\
  D_{xy} & D_{yy}
  \end{bmatrix}
  \]

Let \( \alpha \) be the eigenvalue with larger magnitude and \( \beta \) the smaller.

So \( \alpha = r \beta \)

\[ r = \frac{\alpha}{\beta} \]

\[ r < 10 \]

\[ \frac{\text{Tr}(H)^2}{\det(H)} = \frac{(\alpha + \beta)^2}{\alpha \beta} = \frac{(r \beta + \beta)^2}{r \beta^2} = \frac{(r + 1)^2}{r}, \]

\( (r+1)^2/r \) is at a min when the 2 eigenvalues are equal.

Parity Checker Example

A string of bits has “even parity” if the number of 1’s in the string is even.

- Design a circuit that accepts a bit-serial stream of bits and outputs a 0 if the parity thus far is even and outputs a 1 if odd:

Next we take this example through the “formal design process”. But first, can you guess a circuit that performs this function?
Formal Design Process

“State Transition Diagram”
- circuit is in one of two “states”.
- transition on each cycle with each new input, over exactly one arc (edge).
- output depends on which state the circuit is in.

State Transition Table:

<table>
<thead>
<tr>
<th>present state</th>
<th>OUT</th>
<th>IN</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVEN</td>
<td>0</td>
<td>0</td>
<td>EVEN</td>
</tr>
<tr>
<td>EVEN</td>
<td>0</td>
<td>1</td>
<td>ODD</td>
</tr>
<tr>
<td>ODD</td>
<td>1</td>
<td>0</td>
<td>ODD</td>
</tr>
<tr>
<td>ODD</td>
<td>1</td>
<td>1</td>
<td>EVEN</td>
</tr>
</tbody>
</table>

Invent a code to represent states:
Let 0 = EVEN state, 1 = ODD state

<table>
<thead>
<tr>
<th>present state (ps)</th>
<th>OUT</th>
<th>IN</th>
<th>next state (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Derive logic equations from table (how?):
OUT = PS
NS = PS xor IN
Formal Design Process

Logic equations from table:
OUT = PS
NS = PS xor IN

• Circuit Diagram:
  - XOR gate for NS calculation
  - DFF to hold present state
  - no logic needed for output in this example.