PB99-169328

CAIT REPORT NO. 6

CAIT

CENTER FOR ADVANCED INFRASTRUCTURE
AND TRANSPORTATION
DEPT. OF CIVIL & ENV. ENG—RUTGERS UNIVERSITY

4D DRIVE-THROUGH VISUALIZATION OF I-280 FOR REVIEW OF PROPOSED SIGNING

Gary R. Consolazio
(CAIT Principal Investigator)

CAIT Task Order Number 48—CAIT Study Number 6

Arthur (Mike) Roberts
(NJDOT Project Manager)

October 1998

CAIT—Center for Advanced Infrastructure & Transportation
Rutgers University Dept. of Civil & Environmental Engineering
623 Bowser Road, Piscataway, NJ 08854

Phone: (732) 445-2232
Fax: (732) 445-0577

Web: www.civeng.rutgers.edu
Email: cait@civeng.rutgers.edu

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FHWA/NJ-99-002-CAIT6		

4. Title and Subtitle	5. Report Date
4D DRIVE-THROUGH VISUALIZATION OF I-280 FOR REVIEW OF PROPOSED SIGNING	October 1998
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Gary R. Consolazio	CAIT REPORT NO.6
9. Performing Organization Name and Address	10. Work Unit No.
CAIT - Center for Advanced Infrastructure Technology Rutgers University Dept of Civil & Environmental Engineering 623 Bowser Road Piscataway, NJ 08854 (732)445-2232, FAX(732)445-0577	11. Contract or Grant No.
12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
N J Dept of Transportation CN 600 Trenton, NJ 08625 Federal Highway Administration U.S. Department of Transportation Washington, D.C.	Final Report 1/97 - 10/98
	14. Sponsoring Agency Code

15. Supplementary Notes

Prepared in cooperation with the New Jersey Department of Transportation and USDOT Federal Highway Administration

16. Abstract

The primary objective of this work was to produce a simulated 4D Drive-through of a portion of highway (I-280 through Newark, NJ) for which proposed signing for the Performing Arts Center, Penn Station and other traffic generators in the City of Newark had to be reviewed. In addition to reproducing the existing road in a 3D model, a plan for an exit ramp connecting I-280 to Route 21 in Newark, NJ was also constructed in the 3D model. Using a simulated 4D drive-through, reviewers from NJDOT Traffic Engineering and a public task force were able to see the signing from the point of view of a driver traveling along the roadway at normal traffic speeds. The drive-through visualization was used to help identify potential problems regarding excessive density of signing and inadequate sight distances for signing prior to physical installation of the signs. The 3D geometrical model of the roadway, shoulders, signs, intersecting bridge structures, and other objects of significance were constructed based on available data such as GPS and video log information.

The modeling and visualization processes developed during this project demonstrated that 3D modeling and 4D visualization can be very effectively used for both design purposes and for public presentation purposes. The project also demonstrated that 3D models and 4D visualizations can be created in relatively short time frames.

Recommendations for NJDOT improvements to 3D data generation and 3D data handling software included enhancements to the ARAN van, integrating elevation data in the NJDOT GIS, and faster 3D model generation tools at NJDOT.

17. Key Words	18. Distribution Statement
3D modeling 4D visualization Guide sign placement Driver review	No restrictions

19. Security Classif (of this report)	20. Security Classif. (of this page)	21. No of Pages	22. Price
Unclassified	Unclassified	41	

Form DOT F 1700.7 (8-69)

DISCLAIMER STATEMENT

The contents of this report reflect the views of the author who is responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the views or policies of Rutgers University, the New Jersey Department of Transportation, or the Federal Highway Administration. This report does not constitute a standard, specification, or regulation.

NOTICE

Neither the New Jersey Department of Transportation nor the Federal Highway Administration endorse products or manufacturers. Trade or Manufacturer's names appear herein solely because they are considered essential to the object of this report.

PROJECT SUMMARY

The primary objective of this work was to produce a simulated 4D drive-through of a portion of highway (I-280 through Newark, NJ) for which proposed traffic-generator signing had to be reviewed. A 4D visualization was produced that combined 3D geometry rendering with the time element. Using the simulated drive-through, reviewers (from NJDOT traffic-engineering) were able to see the signing from the point of view of a driver traveling along the roadway at normal traffic speeds. Potential problems regarding excessive density of signing, inadequate sight distances for signing, etc. could then be identified prior to installation of the signs. The 3D geometrical model of the roadway, shoulders, signs, intersecting bridge structures, and other objects of significance was constructed based on available data such as GPS and video log information.

In addition to the I-280 visualization, a second visualization of the exit ramp connecting I-280 to Route 21 in Newark, NJ was also constructed. However, whereas GPS data was used heavily in the modeling of I-280, no such data was available for the exit ramp because it was in the early stages of construction at that time that the visualization was being prepared. Therefore, the 3D model of the Rt. 21 ramp had to be built using only construction plans. The goal of the Rt. 21 ramp visualization was to visualize what the project might look like once construction was completed. In general, this type of visualization can be used during public hearings to more clearly educate and inform the public regarding the impact of proposed construction projects.

Once the 3D geometrical models for this project were created, simulated 4D (3D space + time) drive-throughs of the scenes, including proposed signing, ramps, etc. were constructed. The drive-throughs simulated the point of view of a driver moving along the roadway at a typical traffic speed. The time element of the simulation, i.e. the speed at which the driver moves through the scene, was an important factor in reviewing the proposed signing for I-280. To ensure that the time element was properly represented, each simulation was generated as a series of movie "frames" which were then combined to form a complete movie. Each frame in the movie was constructed by moving the driver's point of view to a new position in the 3D model and then rendering the entire scene including perspective projections, hidden surface removal (necessary to evaluate sight-distance checks), color, lighting, shading, and texture mapping. Texture mapping was used to render the signs and to add realism to the simulation.

Once all of the frames were generated, they were compressed into digital "software" movies that could be easily played back at the correct speed, or which could be used to produce video tapes of the simulation. The modeling and visualization processes developed during this project demonstrated that 3D modeling and 4D visualization can be very effectively used for both design purposes (e.g. traffic engineering design evaluation of proposed signing) and for public presentation purposes. The project also demonstrated that 3D models and 4D visualizations can be created in relatively short time frames.

ACKNOWLEDGEMENTS

The author wishes to thank the New Jersey Department of Transportation (NJDOT) for funding the research described herein.

TABLE OF CONTENTS

Project Summary	i
Acknowledgements	ii
1. Objectives	1
2. Obtaining 3D Alignment Data for I-280	1
3. Obtaining 3D Alignment Data for the Rt. 21 Ramp	4
4. Constructing the 3D Roadway Models	5
5. Integrating Signs Into the 3D Models	7
6. Rendering the 3D Models and Generating Frame Data	9
7. Converting Frame Data into 4D Drive-Throughs	11
8. Implementable Results	12
9. Recommendations for Future Directions	14
Appendix-A Mesh Generation Software	15
Appendix-B Sign Texture Maps for I-280 and Rt. 21 Ramp	29

1. OBJECTIVES

The objective of this project was to produce a simulated 4D drive-through of a portion of highway (interstate I-280 eastbound through Newark, NJ) for which proposed traffic-generator signing had to be reviewed. A 4D visualization was produced that combined 3D geometry rendering with driver motion. Using the simulated drive-through, reviewers at the NJDOT traffic division were able evaluate the proposed signing from the point of view of a driver traveling along the roadway at normal traffic speeds. Potential problems regarding excessive density of signing, inadequate sight distances for signing, etc. could then be identified prior to installation of the signs.

A secondary objective of this project was to create a visualization of the exit ramp connecting I-280 to Route 21 in Newark, NJ. The goal of producing the Rt. 21 ramp visualization was to visualize what the project might look like once construction was completed. Also because the ramp did not exist at the time that the visualization was being prepared, the 3D modeling had to be based entirely on construction plans. Thus, this aspect of the project was aimed at evaluating the feasibility of using 3D modeling and 4D visualization to visualize what a new or proposed construction project might look like upon completion. In general, this type of visualization could be used during public hearings to more clearly educate and inform the public regarding the impact of proposed construction projects.

2. OBTAINING 3D ALIGNMENT DATA FOR I-280

In order to generate a 4D drive-through visualization of the section of I-280 of interest in this project, a 3D geometrical model of the roadway, shoulders, intersecting bridge structures, and other objects of significance first had to be constructed. The 3D model defines the

geometrical characteristics of the roadway (horizontal and vertical alignment), as well as the positions of objects such as signs, bridges, and barriers, relative to the position of the roadway.

The 3D model was constructed based on available data such as GPS and video log information provided by NJDOT. Data were collected from various digital, video and paper sources. GPS data provided by NJDOT consisted of latitude, longitude, and elevation data from an instrumented ARAN van (used as part of NJDOT's pavement system) that was driven along the segment of roadway of interest. Latitude and longitude data from the van were then converted into UTM (universal transverse mercator) coordinates for use in constructing a 3D reference line for the roadway. The 3D reference line was typically the centerline of the rightmost lane of the roadway and was used as the basis for positioning all other objects in the model. Video log (see Figure 1.) data recorded by the van provided additional information that was used to verify the geometry roadway once the 3D model that was constructed. The video log also provided valuable information regarding the vertical alignment of the roadway.

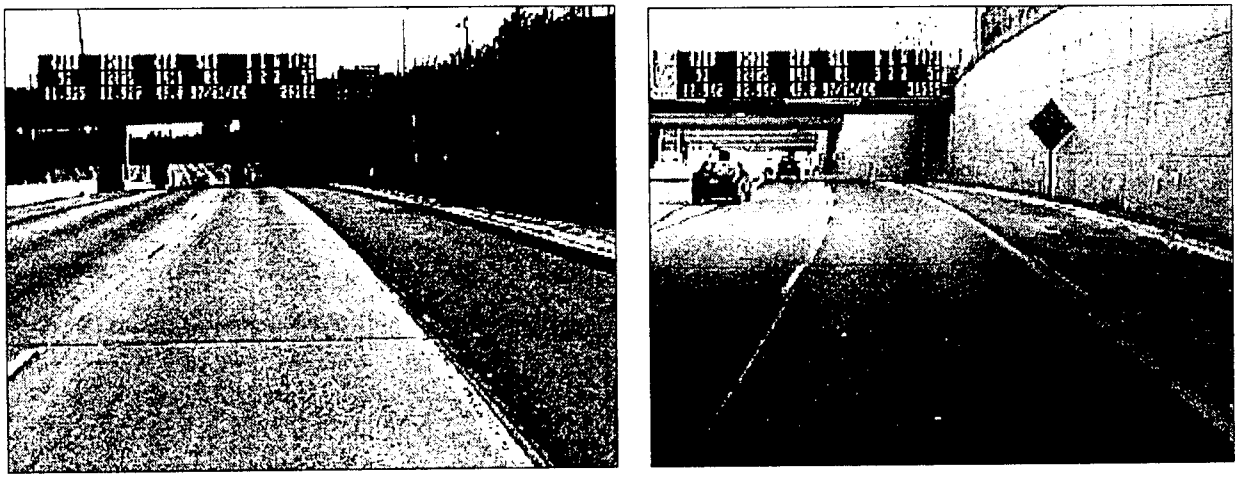


Figure 1. Images from the Video Log

Although the latitude and longitude data provided by the NJDOT ARAN van were very accurate, the elevation data were found to have significant error. Discussions with NJDOT personnel revealed that this was due to a limitation of the instrumentation package onboard the

ARAN van. Although the ARAN instrumentation package *can be* configured to provide very accurate elevation data, the particular version of the ARAN van that NJDOT owns does not have this accuracy. This limitation of the onboard instrumentation package was made to reduce the cost of the van.

To work around the elevation data problem, the elevation data from the ARAN instrumentation had to be supplemented with data from bridge plans and the ARAN video log. Bridge elevations can be used to establish the elevations at key locations along the roadway. The ARAN video log also proved quite useful for adjusting the elevations of the roadway reference line. Once a preliminary 3D roadway model was generated, the Vista visualization package (described later in this report) could be used to render the roadway at particular locations. The rendered scene could then be checked against images from the video log (corresponding to the same milepost position along the roadway) to either verify the vertical alignment of the roadway or to adjust it. If adjustments were necessary, they could be made to the 3D reference line data. The 3D roadway model could then be rebuilt using the revised reference line data and the 3D mesh generation tools described later in this report.

It should be noted that upgrading the instrumentation package onboard the ARAN van—so that it can produce accurate elevation data—would greatly enhance the usefulness of the ARAN van for 3D model generation. Having accurate elevation data that are linked directly to the latitude and longitude data would substantially reduce the time and cost involved in constructing 3D roadway models. Integrating latitude and longitude data from one source (e.g. the ARAN van) with elevation data from other sources (bridge elevations, video log), as was required in this project, requires significant effort. This integration effort could be eliminated by

having the ARAN van produce elevation data that is already linked to the latitude and longitude data being used to construct the 3D roadway reference line.

3. OBTAINING 3D ALIGNMENT DATA FOR THE RT. 21 RAMP

The Rt.21 exit ramp was *under construction* at the time of this project and therefore the NJDOT ARAN van could not be used to obtain 3D alignment data for purposes of building a 3D reference line. Instead, all geometrical data were taken from construction plans (see Figure 2) furnished by NJDOT for the project. A 3D reference line for the ramp was then built using the horizontal and vertical layout data contained in the construction plans. The positions of various roadside features near the Rt. 21 ramp, such as landcaping and on-ramps leading to I-280, were also taken from the plans.

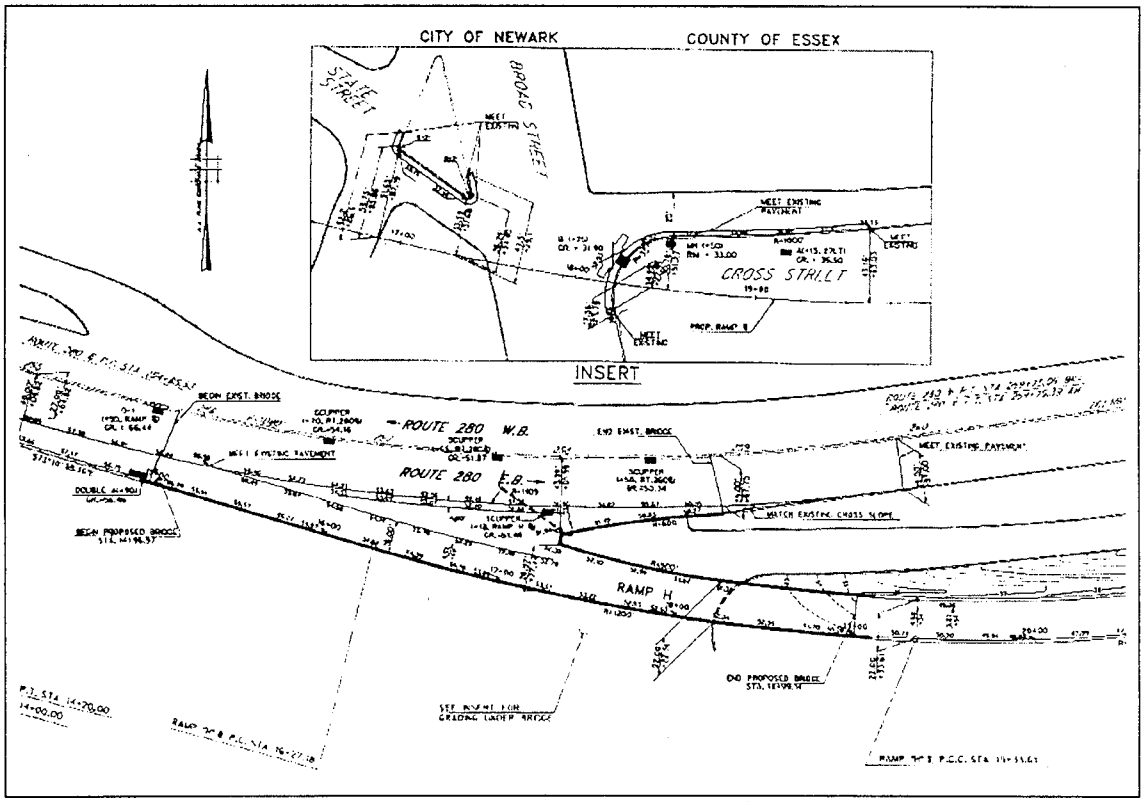


Figure 2. Sample Construction Plan for Rt. 21 Ramp

4. CONSTRUCTING THE 3D ROADWAY MODELS

Using 3D reference lines (constructed using GPS data, video logs, and construction plans) complete 3D models of I-280 eastbound through Newark and of the Rt. 21 exit ramp were generated. The total length of the I-280 segment modeled was approximately four miles. In addition to modeling the roadway, barriers, bridges, and sign structures, on ramps and landscaping were also represented in the 3D models. This task took considerable effort since accurate modeling of the positions and size of these features—especially signs and bridges—was important in ensuring that the model could be used to evaluate aspects of the roadway such as sight-distances.

To assist in the creation of the 3D models from the 3D reference line data, a special semi-manual "meshing" program was written that would create meshes of objects and position them at specified locations relative to the reference line (The source code for this program is given in Appendix A of this report.) In this context, a "mesh" means a collection of surfaces that, when taken together as a whole form a 3D representation of an object. For example, the roadway itself was represented as a large mesh of approximately-rectangular surfaces (see Figure 3). Bridge girders, bridge piers, and concrete barriers in the model are other examples of objects in the 3D model that were represented by collections of 3D surface meshes (see Figure 4). The meshing generator takes as input the following items

- X,Y,Z centerline (reference line) data
- Object dimensions (e.g. width of the roadway, geometry of a sign bridge, etc.)
- Meshing density (quantity of surfaces to be used to represent the meshed object)

and produces as output a fully described (coordinate, connectivity, etc.) collection of surfaces describing the shape of the object generated (meshed).

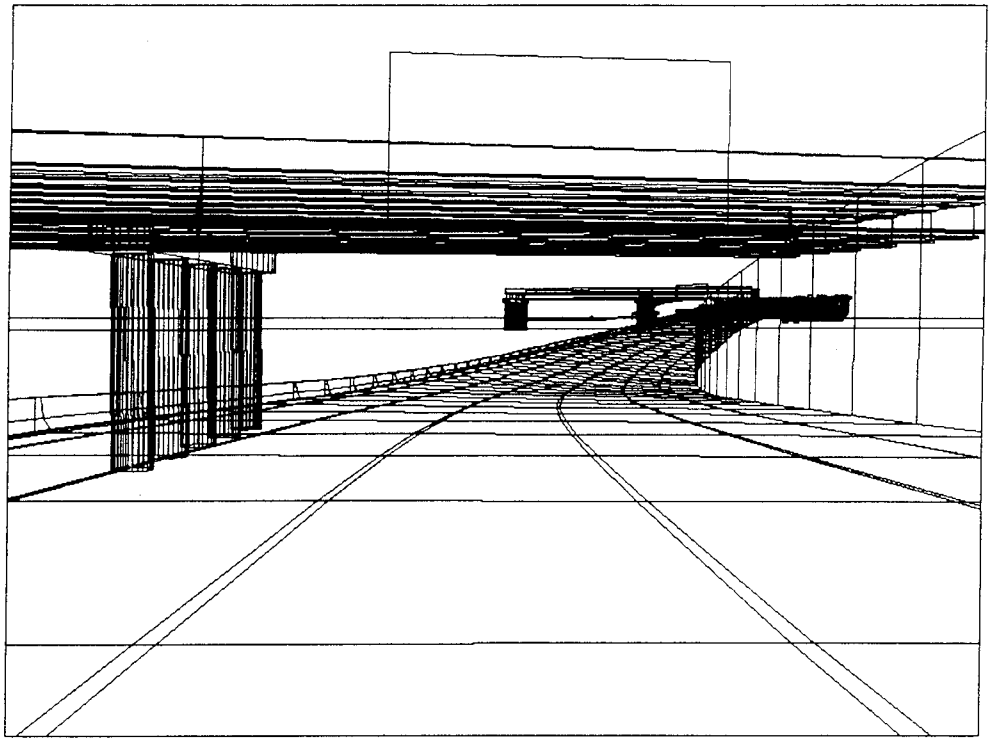


Figure 3. Wire-Frame View of Roadway, Bridge, Sign, and Retaining Wall Mesh Created by Meshing Program

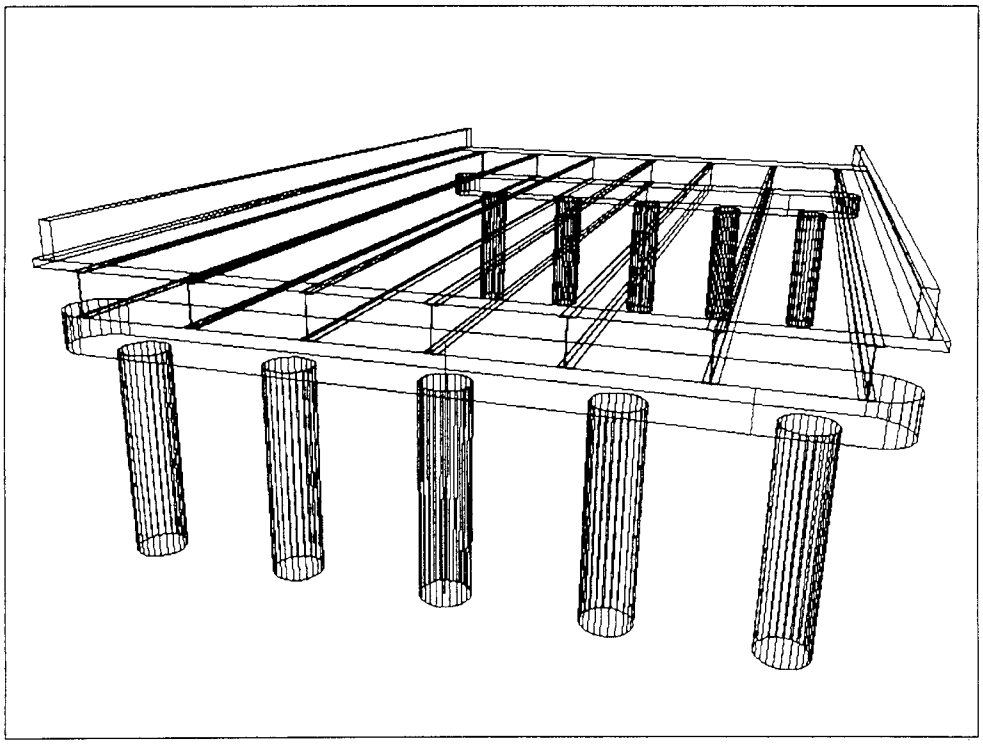


Figure 4. Wire-Frame View of Bridge and Piers Created by Meshing Program

The mesh generation program is considered to be "semi-manual" because it is not a user friendly, graphically based program. Instead it requires a certain degree of manual data preparation before it can be used to generate the coordinate and surface data needed for 3D modeling. It was outside the scope of this project to develop a fully functional and user friendly mesh generation program. It would, however, be very beneficial to develop a fully functional, graphically based (e.g. "Windows style") mesh generation program that has features built into that facilitate rapid development of roadway (and related) meshes.

5. INTEGRATING SIGNS INTO THE 3D MODELS

Once the 3D geometrical roadway models (including bridges, barriers, walls, ramps, etc.) was created, the proposed signing was merged with the 3D roadway models. The I-280 sign faces were obtained from NJDOT in CAD (MicroStation) format and converted for use in the drive-through visualization. The sign faces for the NJPAC (New Jersey Performing Arts Center) signs on the Rt. 21 exit ramp were obtained from the consulting firm that was developing the signs faces for NJDOT for the ramp construction project.

Position and size data for the proposed signing was taken from the plans for the proposed signing. The MicroStation file format was used as the source format for the I-280 signs in this project because NJDOT uses MDL (MicroStation Development Language) programs to generate signage for signage projects. Thus, sign faces are generated directly within MicroStation based programs and then must be imported into the 3D roadway model for final rendering.

Rectangular sign objects, having the correct size, shape, and position relative to the roadway, were merged with the 3D roadway model. These rectangular objects served as surfaces onto which the sign images would be *texture mapped* (digitally painted). Objects representing

sign-posts and sign support structures (e.g. sign bridges and cantilever supports) of the type indicated on the proposed plans were also merged into the 3D model (see Figure 5).

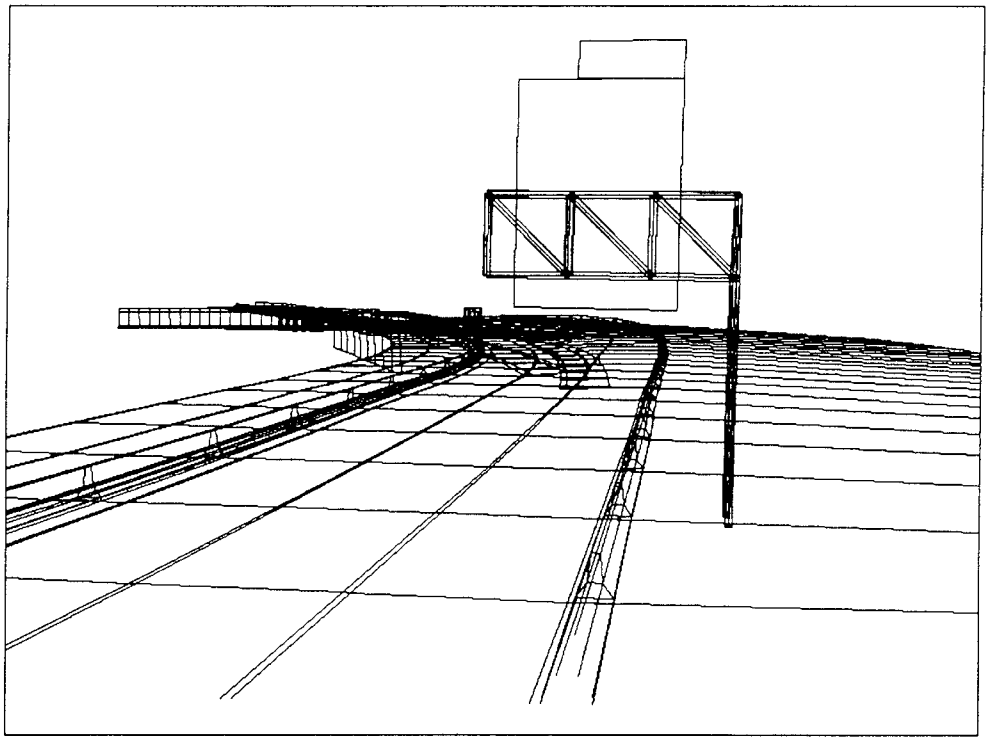


Figure 5. Wire-Frame View of Rectangular Sign Surfaces and Sign Support Structure

The actual sign faces were then added to the model by converting the MicroStation CAD files (see Figure 6) provided by NJDOT into raster based image files. The raster images were then applied (using a 3D rendering technique called *texture mapping*) to sign objects in the 3D model. (The texture maps used to model the sign faces for I-280 and for the Rt. 21 ramp are presented in Appendix B of this report.)

The process of converting the MicroStation files into raster format, and then applying the raster images to objects within the 3D model using texture mapping, was found to be the most rapid method of integrating the proposed signage into the 3D model. It also allowed rendering

techniques such as perspective transformations and lighting to be made directly to the signs since the signs were actually texture mapped 3D objects within the overall model.

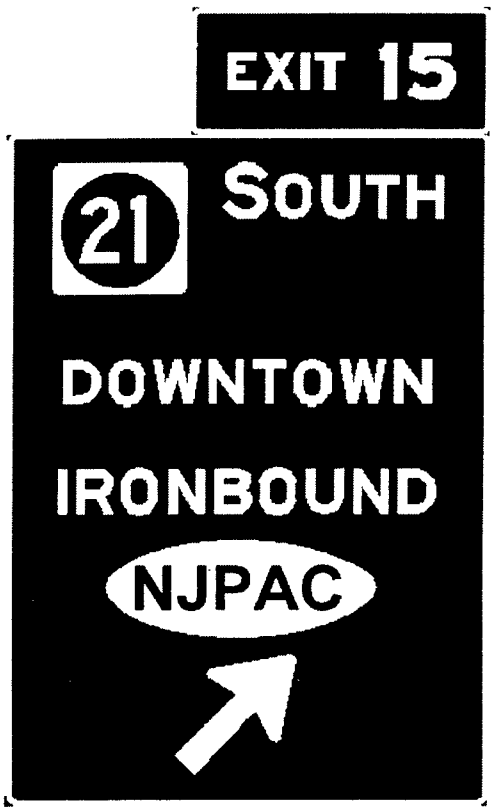


Figure 6. Example Signs to be Integrated into 3D Model
Using Texture Mapping

6. RENDERING THE 3D MODELS AND GENERATING FRAME DATA

The 4D drive-through simulations were created by defining complete viewer (driver) motion paths along the roadways in the 3D models and then moving the point of view of the drivers along those motion paths in small increments of carefully computed size. In each case, i.e. the I-280 simulation and the Rt. 21 ramp simulation, the entire 3D model was rendered at thousands of points along the motion path which resulted in thousands of "frame" images (similar to frames of a movie). A 3D/4D visualization code called *Vista*, previously developed by the principal investigator and enhanced for this study, was used to perform all of the rendering

needed to generate the frame images for this project. Vista is a software package that performs rendering of 3D objects/scenes and features the rendering options listed below.

- Wire-frame, hidden-line, and surface rendering
- Coloring of objects
- Texture mapping of images onto objects (e.g. for modeling of signs)
- Lighting effects
- Extrusion of objects (e.g. for modeling bridge girders, piers, etc.)
- Drive-through (fly-through) animation based on defined motion paths
- Automatic generation and export of "movie frames" for defined motion paths

In order to generate a 4D drive-through visualization, using the frames rendered by Vista, a motion path along the roadway has to be constructed. This was performed by using the same X,Y,Z reference data that was previously used to construct the 3D models. Also, the same meshing tool (software) used to create the 3D models was again utilized to create motion path data for the drive-through animations. Since the "driver" is being moved along the roadway (i.e. through the 3D model) in small but finite size steps, the size of these steps had to be determined. If these had simply been a generic animations, then the size of these steps along the roadway would have been arbitrary—any reasonable step size that produced a smooth animation would have been acceptable.

However, the visualization needed for the I-280 portion of this project was more than a simple animation. The drive-through visualization had to move the viewer through the scene (along the roadway) at *proper driving speeds* so that the engineers reviewing the visualization would see the signing in the same way that an actual driver would. Therefore, the size of the shifts that were used to "move" the viewer along the roadway were computed to produce the correct rate of movement through the model in the final 4D drive-through. In order to compute the required shift size, both the desired driver speed and the final visualization frame-rate playback speed had to be known. For this project, a driver speed of 55 mph and a drive-through

playback frame rate (number of frames that can be displayed per second during viewing of the visualization) of 15 fps (frame per second) were chosen.

A speed of 55 miles per hour is equivalent to a speed of 80.7 feet per second. Therefore, if the visualization will be played back at a frame rate of 15 frames per second, the required driver shift size is (80.7 feet/sec) / (15 frames/second) = 5.38 feet/frame. In other words, the position of the driver moving along the roadway must be moved in increments of 5.38 feet. After each movement, the entire roadway scene is rendered from the point of view of the driver, then the rendered scene is written out to an image file. This creates a single frame of the final drive-through movie that will be produced. For a four mile segment of I-280, the number of frames required, moving in shifts of 5.38 feet, is approximately 4000. The 4000 frames were rendered and written automatically by Vista by moving along the defined motion path. Example frames generated in this manner for the I-280 visualization are shown in Figure 7.

For the Rt. 21 ramp visualization, the speed of the driver's motion along the ramp was less important. This was due to the fact that this visualization was only being used to evaluate the probable appearance of the ramp after construction was completed. Therefore, the rate of motion along the exit ramp was chosen with only "animation smoothness" in mind. Example frames generated for the Rt. 21 visualization are shown in Figure 8.

7. CONVERTING FRAME DATA INTO 4D DRIVE-THROUGHS

The frame image files generated by Vista were subsequently compiled and compressed into movies in both MPEG (motion pictures expert group) and AVI (video for windows) format. The MPEG movie was produced using both freely available and commercial MPEG video compression software packages. To compile the AVI format movies, the commercial product *Adobe Premiere* was used.

It was found that AVI format files, while larger in size than MPEG files, provided higher quality images (in terms of visual clarity of signs and smoothness of animation) than did MPEG. As a final step, the MPEG and AVI movies were both transferred to video tape for purposes of easier viewing using a video cassette player. However, some image degradation does occur in going from digital (e.g. AVI) to VHS format. Therefore, if possible, it is preferable to view the drive-through visualizations by using a computer to display the AVI files.

8. IMPLEMENTABLE RESULTS

The drive-through visualization created for this project was used by NJDOT traffic engineers to review the proposed signing of a segment of I-280 in Newark, NJ so that possible sight-distance problems could be detected and fixed prior to construction. In addition, the Rt. 21

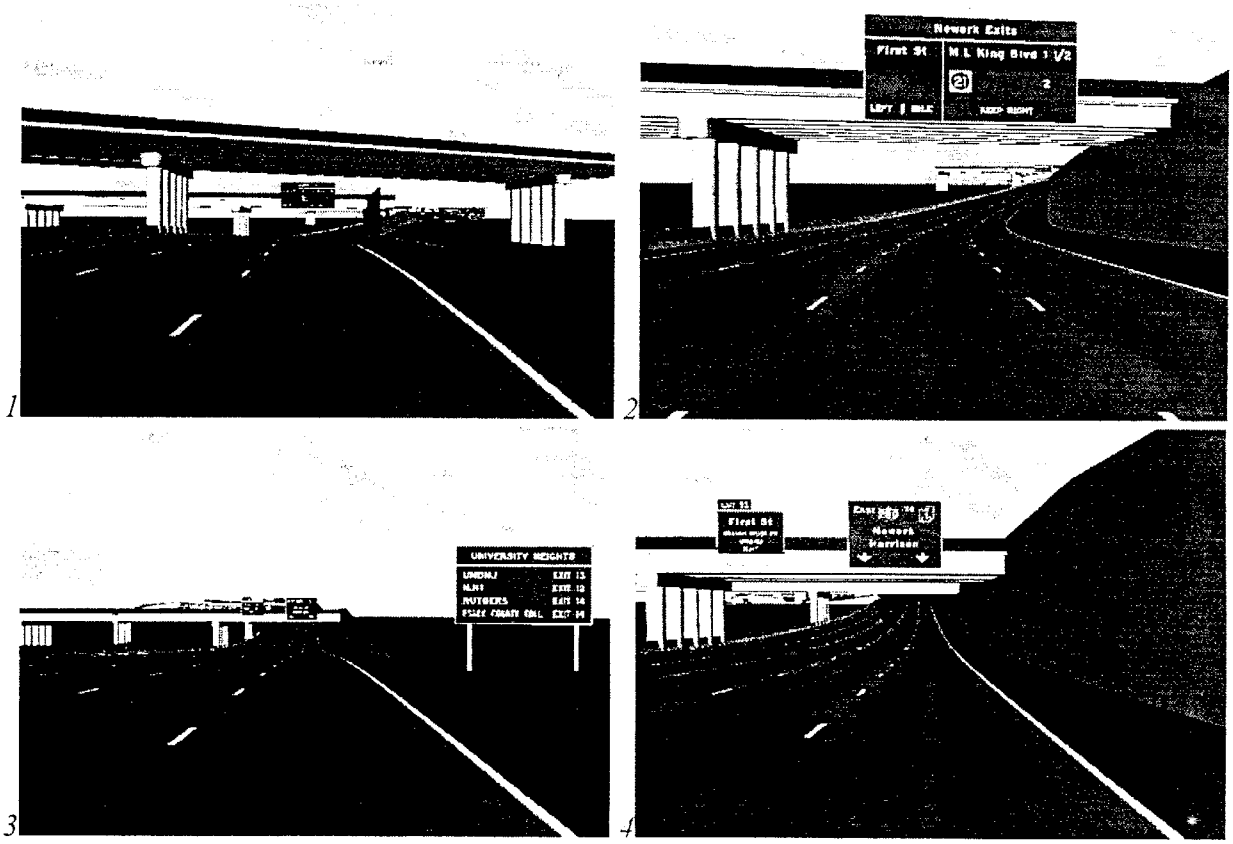


Figure 7. Example Frames from the I-280 Visualization

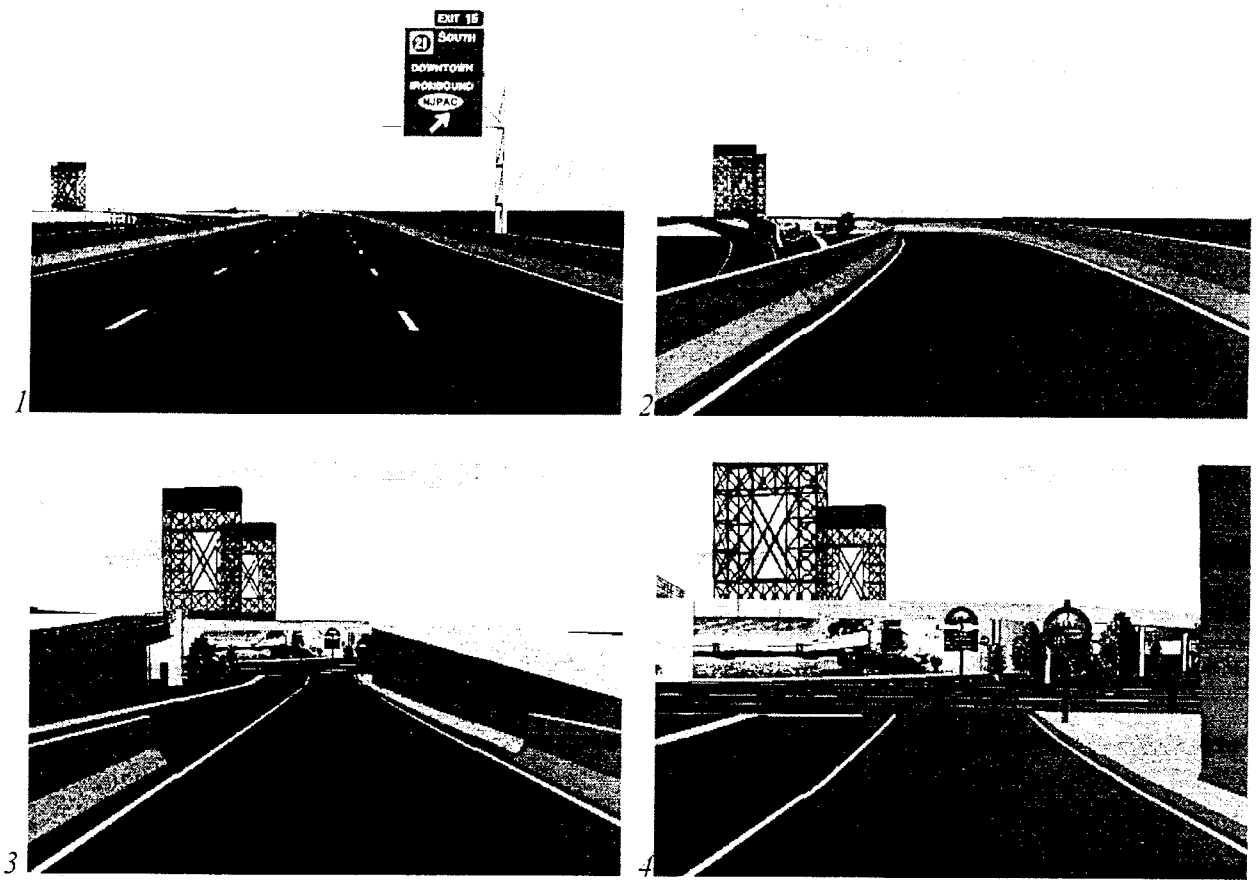


Figure 8. Example Frames from the Rt. 21 Ramp Visualization

ramp being constructed at the time was visualized (before it was complete) based solely on construction plans, clearly demonstrating the potential of this type of presentation tool for public hearings and similar presentation scenarios.

The visualization process, software tools, and simulations developed during this project have demonstrated the feasibility of using this type of visualization for pre-construction evaluation of signing projects to avoid costly design changes later during the construction process.

9. RECOMMENDATIONS FOR FUTURE DIRECTIONS

The results of this project have shown that the use of 3D modeling and 4D visualization for the direct review of proposed highway signing is both highly feasible and very desirable. However, in order to make the use of visualization more routine, better 3D data collection should be pursued and better 3D model development software tools must be developed.

Improvements to the 3D data collection process could be addressed in several manners. An enhanced instrumentation package added to the NJDOT ARAN van would reduce the effort involved constructing 3D models for existing roadways. Also, integrating elevation data in the NJDOT GIS (geographic information system) would also prove very useful. At present, roadway elevation data are not available in the NJDOT GIS. Integration of such data would enhance the usefulness of the GIS for 3D roadway model construction.

Improved 3D model generation tools must also be created if routine use of this technology is to occur. Traffic engineers and planners must have the ability to rapidly construct 3D models, establish drive-through paths, and create rendered visualizations with a minimum of time and effort. In addition to facilitating rapid model development for use in the traffic design area, the creation of such a software tool would also facilitate the use of this technology for generating visualizations for public hearings. Such visualizations, which would include not only proposed signing but also proposed roadway additions and alterations, could be used to very clearly present to the public the impact of a proposed project on a community or area.

APPENDIX A – MESH GENERATION SOFTWARE

This appendix contains the code listing for the "semi-manual" mesh generation software that was written for this project. The code listed below makes use of other utility libraries that were previously written by the author of this report and which are not included herein.

```
/*
 * meshobj.c -- generate mesh data for objects such as roadway, bridges, etc.
 * --gary consolazio 06-mar-97 grc@civeng.rutgers.edu
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include "read.p"
#include "eqn.p"
#include "alias.p"
#include "util.p"

typedef double real;

typedef struct {
  real x;                                                    /* x-coordinate */
  real y;                                                    /* y-coordinate */
  real z;                                                    /* z-coordinate */
} coor_t;

typedef struct {
  real x;                                                    /* x-coordinate */
  real y;                                                    /* y-coordinate */
  real z;                                                    /* z-coordinate */
  real len;                                                 /* vector length */
} vect_t;

typedef struct {
  int beforeTwo;  /* index of data point lying 2 places before position "s" */
  int before;              /* index of data point lying before position "s" */
  int after;                /* index of data point lying after position "s" */
  int afterTwo;    /* index of data point lying 2 places after position "s" */
  real locpos;        /* local position [0.0, 1.0] of "s" between neighbors */
} neighbor_t;

#define strEql(x,y) (strcmp((x),(y))==0)                    /* strings equal */
#define strNeq(x,y) (strcmp((x),(y))!=0)                 /* string not equal */

#define segLength(src,dest) \
(sqrt((dest.x-src.x)*(dest.x-src.x)+ \
      (dest.y-src.y)*(dest.y-src.y)+ \
      (dest.z-src.z)*(dest.z-src.z)))

/* protos */

char *strClone(char *string);
neighbor_t findNeighbor (real s, real *sPath, int numPoint);
coor_t quad3dInterp(real s, real *sPath, neighbor_t neighbor, coor_t *data);
real quad2dInterp(real s, real *sPath, neighbor_t neighbor, real *data);
real quad1dInterp(real s1, real s2, real s3, real f1, real f2, real f3);
vect_t vectorForm ( coor_t i, coor_t j );
vect_t vectorCross ( vect_t v1, vect_t v2 );
real vectorDot ( vect_t v1, vect_t v2 );
vect_t vectorUnit ( vect_t vect );

#define NONE -1

#define ROTATE 1
#define LOCATE 2

#define DEG2RAD 0.017453293

#if 0
#define DEBUG
#endif
```

```
void cleanup_ (void) {} /* application specific cleanup routine */

/*-----------------------------------------------------------------------------
main -- main module
-----------------------------------------------------------------------------*/

int
main(int argc, char *argv[])
{
  char *filename;
  int stat,err;                /* return and error status codes              */
  char buf[80];
  char *errMsg;
  real postStart;
  real postEnd;
  int numDiv;
  int nodeStart;
  int elemStart;
  char *filenameRef;
  char *filenameSlope;
  real sectHorz;
  real sectLeft;
  int sectAdj;
  real *u, *v, *w;
  int numSectPnt;
  int idx;
  int idxDiv,idxSect;
  FILE *f;
  int numRefPnt;
  coor_t *ref;
  real *refList;
  int numSlopePnt;
  real *slope;
  real *slopeList;
  real postStartRefData, postEndRefData;
  real refLength;
  real refDelta;
  real refStart, refEnd;
  real refCur;
  real offset;
  coor_t iCoor, jCoor, kCoor;
  neighbor_t neighbor;
  vect_t rVect, sVect, tVect;
  real postStartSlopeData;
  real postEndSlopeData;
  real slopeLength;
  real curSlope;
  real slopeOffset;
  int nodeNum;
  int elemNum;
  int side;
  coor_t coor;
  real refCurPost;
  int n1,n2,n3,n4;
  int numTokens;

  if( argc != 2 ){
    printf("usage: meshroad filename \n");
    exit (1);
  }
  filename = argv[1];

  EnterFile(filename);
  AliasBuild("symbol");

  EnterBlock("control", "block");

  /*---------------------------------------*/
```

```
  /* read mesh generation control parameters */
  /*-----------------------------------------*/

  errMsg = "(missing control parameter)\n\
  \n Could not read the control parameter \"%s\".\
  \n";

  if(!ReadData(&err, "post_start", "d", &postStart))
    Error ( "main", errMsg, "post_start");

  if(!ReadData(&err, "post_end", "d", &postEnd))
    Error ( "main", errMsg, "post_end");

  if(!ReadData(&err, "num_div", "i", &numDiv))
    Error ( "main", errMsg, "num_div");

  if(!ReadData(&err, "node_start", "i", &nodeStart))
    Error ( "main", errMsg, "node_start");

  if(!ReadData(&err, "elem_start", "i", &elemStart))
    Error ( "main", errMsg, "elem_start");

  if(!ReadData(&err, "ref_data", "a", buf))
    Error ( "main", errMsg, "refData");
  else
    filenameRef = strClone(buf);

  if(!ReadData(&err, "slope_data", "a", buf))
    Error ( "main", errMsg, "slope_data");
  else
    filenameSlope = strClone(buf);

  if(!ReadData(&err, "sect_horz", "d", &sectHorz))
    sectHorz = 0.0; /* default if not found */
/* Error ( "main", errMsg, "sect_horz"); */

  if(!ReadData(&err, "sect_adj", "a", buf))
    Error ( "main", errMsg, "sect_adj");
  else{
    if(strEql(buf, "rotate"))
      sectAdj = ROTATE;
    else if(strEql(buf, "locate"))
      sectAdj = LOCATE;
    else
      Error ( "main", errMsg, "sect_adj");
  }

  printf("# post_start = %f \n",postStart);
  printf("# post_end = %f \n",postEnd);
  printf("# num_div = %d \n",numDiv);
  printf("# node_start = %d \n",nodeStart);
  printf("# elem_start = %d \n",elemStart);
  printf("# filenameRef = %s \n",filenameRef);
  printf("# filenameSlope = %s \n",filenameSlope);
  printf("# sect_horz = %f \n",sectHorz);
  printf("# sect_adj = %s \n",(sectAdj==ROTATE)?"ROTATE":"LOCATE");
  printf("#\n");
  ExitBlock("control");

  /*--------------------*/
  /* read section data */
  /*--------------------*/

  EnterBlock("section", "line");
  ReadInquire("num_lines", &numSectPnt);

  u = (real *) Malloc(numSectPnt*sizeof(real));
  v = (real *) Malloc(numSectPnt*sizeof(real));
```

```
    w = (real *) Malloc(numSectPnt*sizeof(real));

    printf("# local section coordinates \n");
    for(idx=0; idx<numSectPnt ; idx++){
      ReadLine();
      ReadInquire("num_tokens", "", &numTokens);
      if(numTokens==2){
        ReadData(&err, "", "dd", &(u[idx]), &(v[idx]));
        w[idx] = 0.0;
      }else{
        ReadData(&err, "", "ddd", &(u[idx]), &(v[idx]), &(w[idx]));
      }
      printf("# %10.5f %10.5f %10.5f \n", u[idx], v[idx], w[idx]);
    }
    printf("#\n");
    printf("# read %d points of section data.\n",numSectPnt);
    printf("#\n");
    ExitBlock("section");
    ExitFile();

    /*---------------------------*/
    /* read reference (xyz) data */
    /*---------------------------*/

    EnterFile(filenameRef);
    AliasBuild("symbol");
    EnterBlock("data", "line");
    ReadInquire("num_lines", &numRefPnt);

    ref = (coor_t *) Malloc(numRefPnt*sizeof(coor_t));
    refList = (real *) Malloc(numRefPnt*sizeof(real));
    for(idx=0; idx<numRefPnt ; idx++){
      ReadLine();
      ReadData(&err, "", "dddd",
        &(ref[idx].x), &(ref[idx].y), &(ref[idx].z), &refList[idx]);
    }
    printf("# read %d points of reference (xyz) data.\n",numRefPnt);
    postStartRefData = refList[0];;
    postEndRefData   = refList[numRefPnt-1];;
    ExitBlock("data");
    ExitFile();

    /* convert absolute mileposts to relative foot-posts */

    refList[0] = 0.0;
    refLength = 0.0;
    for( idx=1 ; idx<numRefPnt; idx++ ){ /* note beginning index is 1 */
      refLength += segLength(ref[idx-1], ref[idx]);
      refList[idx] = refLength;
    }
    printf("# reference path length is %f (ft) = %f (mi).\n",
      refLength, refLength/5280.0);

    printf("#\n");

    /*-----------------*/
    /* read slope data */
    /*-----------------*/

    EnterFile(filenameSlope);
    AliasBuild("symbol");
  /*AliasPrint("symbol",stdout);*/
    EnterBlock("data", "line");
    ReadInquire("num_lines", &numSlopePnt);

    slope = (real *) Malloc(numSlopePnt*sizeof(real));
    slopeList = (real *) Malloc(numSlopePnt*sizeof(real));
    for(idx=0; idx<numSlopePnt ; idx++){
```

```
    ReadLine();
    ReadData(&err, "", "dd", &(slopeList[idx]), &(slope[idx]));
  }
  printf("# read %d points of slope data.\n",numSlopePnt);
  postStartSlopeData = slopeList[0];;
  postEndSlopeData   = slopeList[numSlopePnt-1];

  /* convert absolute mileposts to relative foot-posts */

  slopeList[0] = 0.0;
  for( idx=1 ; idx<numSlopePnt; idx++ ){ /* note beginning index is 1 */
    slopeList[idx] = 5280.0 * (slopeList[idx]-postStartSlopeData);
  }
  slopeLength = 5280.0 * (postEndSlopeData - postStartSlopeData);
  printf("# slope path length is %f (ft) = %f (mi).\n",
    slopeLength, slopeLength/5280.0);
  printf("#\n");
  ExitBlock("data");
  ExitFile();

  if(postStartRefData != postStartSlopeData){
    printf("error: ref and slope data must begin at same mile post.\n");
    exit(1);
  }

  /*----------------*/
  /* generate nodes */
  /*----------------*/

  printf("# generated nodes\n");

  refStart = (postStart - postStartRefData) * 5280.0;
  refEnd   = (postEnd   - postStartRefData) * 5280.0;
  if(numDiv==0)
    refDelta = 0.0;
  else
    refDelta = (refEnd - refStart) / numDiv;

  offset = refLength / (10.0*(numRefPnt-1)); /* approx. offset value */

  nodeNum = nodeStart-1;

  /* generate nodes -- generation is along longitudinal road direction */

  if(sectAdj == ROTATE){

    /*--------------*/
    /* case: ROTATE */
    /*--------------*/

    for(idxSect=0; idxSect<numSectPnt ; idxSect++){
      if(idxSect>0)
        printf("#\n");
      for(idxDiv=0; idxDiv<=numDiv ; idxDiv++){
        /* compute current path position along reference data path */

        refCur = refStart + idxDiv*refDelta;

        refCurPost = (refCur/5280.0)+postStartRefData;

        if(refCur<0 || (refCur+offset)>refList[numRefPnt-1]){
          printf("error: refCur out of range for reference data.\n");
          exit(1);
        }
        neighbor = findNeighbor(refCur, refList, numRefPnt);
        iCoor = quad3dInterp(refCur, refList, neighbor, ref);

        neighbor = findNeighbor(refCur+offset, refList, numRefPnt);
```

```
        jCoor = quad3dInterp(refCur+offset, refList, neighbor, ref);

        kCoor = iCoor;
        kCoor.z += offset;

        /* get slope data at this point */

        if(refCur<0 || (refCur+offset)>slopeList[numSlopePnt-1]){
          printf("error: refCur out of range for slope data.\n");
          exit(1);
        }

        neighbor = findNeighbor(refCur, slopeList, numSlopePnt);
        curSlope = quad2dInterp(refCur, slopeList, neighbor, slope);

        /* setup local 3d coor system along road direction */

        rVect = vectorForm(iCoor, jCoor);
        tVect = vectorForm(iCoor, kCoor);
        sVect = vectorCross(tVect, rVect);
        tVect = vectorCross(rVect, sVect);

        rVect = vectorUnit(rVect);
        sVect = vectorUnit(sVect);
        tVect = vectorUnit(tVect);

        /* modify local 3d coor system for specified cross slope */

        slopeOffset = tan(DEG2RAD*curSlope);
        tVect.x += -slopeOffset*sVect.x;
        tVect.y += -slopeOffset*sVect.y;
        tVect.z += -slopeOffset*sVect.z;
        tVect = vectorUnit(tVect);
        sVect = vectorCross(tVect, rVect);
        sVect = vectorUnit(sVect);

        nodeNum++;
        coor = iCoor;

        coor.x += -u[idxSect]*sVect.x;
        coor.y += -u[idxSect]*sVect.y;
        coor.z += -u[idxSect]*sVect.z;

        coor.x += v[idxSect]*tVect.x;
        coor.y += v[idxSect]*tVect.y;
        coor.z += v[idxSect]*tVect.z;

        coor.x += -w[idxSect]*rVect.x;
        coor.y += -w[idxSect]*rVect.y;
        coor.z += -w[idxSect]*rVect.z;

        printf("%8d, %10.2f, %10.2f, %10.2f   #  mile %8.5f \n",
          nodeNum, coor.x, coor.y, coor.z, refCurPost);
      }
    }
  }else if(sectAdj == LOCATE){

    /*--------------*/
    /* case: LOCATE */
    /*--------------*/

    for(idxSect=0; idxSect<numSectPnt ; idxSect++){
      if(idxSect>0)
        printf("#\n");
      for(idxDiv=0; idxDiv<=numDiv ; idxDiv++){
        /* compute current path position along reference data path */

        refCur = refStart + idxDiv*refDelta;
```

```
refCurPost = (refCur/5280.0)+postStartRefData;

if(refCur<0 || (refCur+offset)>refList[numRefPnt-1]){
  printf("error: refCur out of range for reference data.\n");
  exit(1);
}
neighbor = findNeighbor(refCur, refList, numRefPnt);
iCoor = quad3dInterp(refCur, refList, neighbor, ref);

neighbor = findNeighbor(refCur+offset, refList, numRefPnt);
jCoor = quad3dInterp(refCur+offset, refList, neighbor, ref);

kCoor = iCoor;
kCoor.z += offset;

/* get slope data at this point */

if(refCur<0 || (refCur+offset)>slopeList[numSlopePnt-1]){
  printf("error: refCur out of range for slope data.\n");
  exit(1);
}

neighbor = findNeighbor(refCur, slopeList, numSlopePnt);
curSlope = quad2dInterp(refCur, slopeList, neighbor, slope);

/* setup local 3d coor system along road direction */

rVect = vectorForm(iCoor, jCoor);
tVect = vectorForm(iCoor, kCoor);
sVect = vectorCross(tVect, rVect);
tVect = vectorCross(rVect, sVect);

rVect = vectorUnit(rVect);
sVect = vectorUnit(sVect);
tVect = vectorUnit(tVect);

/* modify local 3d coor system for specified cross slope */

slopeOffset = tan(DEG2RAD*curSlope);
tVect.x += -slopeOffset*sVect.x;
tVect.y += -slopeOffset*sVect.y;
tVect.z += -slopeOffset*sVect.z;
tVect = vectorUnit(tVect);
sVect = vectorCross(tVect, rVect);
sVect = vectorUnit(sVect);

nodeNum++;
coor = iCoor;

/* locate origin of local "located" coordinated system */

coor.x += -sectHorz*sVect.x;
coor.y += -sectHorz*sVect.y;
coor.z += -sectHorz*sVect.z;

/*
 * modify according to local coordinates:
 * +u local -> move in xy plane (at constant z-elevation) to right.
 * +v local -> move in +z direction only
 * +w local -> move in xy plane (at constant z-elevation) backward.
 */

/* remove z-component and renormalize */
sVect.z = 0.0;
sVect = vectorUnit(sVect);
rVect.z = 0.0;
rVect = vectorUnit(rVect);
```

```
            coor.x += -u[idxSect]*sVect.x;
            coor.y += -u[idxSect]*sVect.y;
            coor.z +=  v[idxSect];
            coor.x += -w[idxSect]*rVect.x;
            coor.y += -w[idxSect]*rVect.y;

            printf("%8d, %10.2f, %10.2f, %10.2f    #  mile %8.5f \n",
              nodeNum, coor.x, coor.y, coor.z, refCurPost);
          }
        }
      }
      printf("#\n");

      /*----------------*/
      /* generate elems */
      /*----------------*/

      printf("# generated elements\n");

      elemNum = elemStart-1;
      for(idxSect=0; idxSect<numSectPnt-1 ; idxSect++){
        if(idxSect>0)
          printf("#\n");
        for(idxDiv=0; idxDiv<numDiv ; idxDiv++){
          elemNum++;
          n1 = nodeStart + idxSect*(numDiv+1) + idxDiv;
          n2 = n1 + 1;
          n3 = n2 + numDiv+1;
          n4 = n1 + numDiv+1;

          printf("%5d,  %5d, %5d, %5d, %5d \n", elemNum, n1,n2,n3,n4);
        }
      }
    }

    /*-------------------------------------------------------------------------
    findNeighbor -- find indices of points neighboring path position "s"
    -------------------------------------------------------------------------*/
    neighbor_t
    findNeighbor (real s, real *sPath, int numPoint)
    {
      neighbor_t neighbor;
      int i;
      int found;

      found = 0;

      if(s<=0.0){
        /* extrapolate using first quad */
        neighbor.beforeTwo = NONE;
        neighbor.before    = 0;
        neighbor.after     = 1;
        neighbor.afterTwo  = 2;
        neighbor.locpos    = 0.0;
        found = 1;

      }else if(s>=sPath[numPoint-1]){
        /* extrapolate using last quad */
        neighbor.beforeTwo = numPoint-3;
        neighbor.before    = numPoint-2;
        neighbor.after     = numPoint-1;
        neighbor.afterTwo  = NONE;
        neighbor.locpos = 1.0;
        found = 1;

      }else{
        for( i=1 ; i<numPoint ; i++ ){
```

```
        if(sPath[i] >= s){

          if(i==1){
            neighbor.beforeTwo = NONE;
          }else{
            neighbor.beforeTwo = i-2;
          }
          neighbor.before    = i-1;
          neighbor.after     = i;

          if(i==numPoint-1){
            neighbor.afterTwo  = NONE;
          }else{
            neighbor.afterTwo  = i+1;
          }
          neighbor.locpos = (s-sPath[i-1])/(sPath[i]-sPath[i-1]);
          found = 1;
          break;
        }
      }
    }
    if(found)
      return neighbor;
    else{
      printf("error in \"findNeighbor\" -- could not find neighbors \n");
      exit(1);
    }
}

/*---------------------------------------------------------------------------
quad3dInterp -- do 3d quadratic function interpolation
---------------------------------------------------------------------------*/

coor_t
quad3dInterp(real s, real *sPath, neighbor_t neighbor, coor_t *data)
{
  coor_t coor,coora,coorb;
  real whta, whtb;
  real s1,s2,s3;
  int base;
  int sDelta;

  coora.x=0.0; coora.y=0.0; coora.z=0.0;
  coorb.x=0.0; coorb.y=0.0; coorb.z=0.0;

  if(neighbor.beforeTwo!=NONE){
    /* use quadratic interp */
    base = neighbor.beforeTwo;
    s1 = sPath[base] - s;
    s2 = sPath[base+1] - s;
    s3 = sPath[base+2] - s;
    coora.x = quad1dInterp(
      s1,s2,s3, data[base].x,data[base+1].x,data[base+2].x);
    coora.y = quad1dInterp(
      s1,s2,s3, data[base].y,data[base+1].y,data[base+2].y);
    coora.z = quad1dInterp(
      s1,s2,s3, data[base].z,data[base+1].z,data[base+2].z);
  }else{
    /* use linear interp */
    base = neighbor.before;
    coora.x = data[base].x + neighbor.locpos * (data[base+1].x-data[base].x);
    coora.y = data[base].y + neighbor.locpos * (data[base+1].y-data[base].y);
    coora.z = data[base].z + neighbor.locpos * (data[base+1].z-data[base].z);
  }
  if(neighbor.afterTwo!=NONE){
    /* use quadratic interp */
    base = neighbor.before;
    s1 = sPath[base] - s;
```

```
      s2 = sPath[base+1] - s;
      s3 = sPath[base+2] - s;
      coorb.x = quad1dInterp(
        s1,s2,s3, data[base].x,data[base+1].x,data[base+2].x);
      coorb.y = quad1dInterp(
        s1,s2,s3, data[base].y,data[base+1].y,data[base+2].y);
      coorb.z = quad1dInterp(
        s1,s2,s3, data[base].z,data[base+1].z,data[base+2].z);
    }else{
      /* use linear interp */
      base = neighbor.before;
      coorb.x = data[base].x + neighbor.locpos * (data[base+1].x-data[base].x);
      coorb.y = data[base].y + neighbor.locpos * (data[base+1].y-data[base].y);
      coorb.z = data[base].z + neighbor.locpos * (data[base+1].z-data[base].z);
    }

    /* do weighted averaging */

    sDelta = sPath[neighbor.after] - sPath[neighbor.before];
    if(sDelta == 0.0){ /* just in case this happens ... */
      whta = 0.5; whtb = 0.5;
    }else{
      whta = 1.0 - (s-sPath[neighbor.before])/sDelta;
      whtb = 1.0 - whta;
    }
    coor.x = whta*coora.x + whtb*coorb.x;
    coor.y = whta*coora.y + whtb*coorb.y;
    coor.z = whta*coora.z + whtb*coorb.z;

    return coor;
}

/*---------------------------------------------------------------------------
quad2dInterp -- do 2d quadratic function interpolation
---------------------------------------------------------------------------*/

real
quad2dInterp(real s, real *sPath, neighbor_t neighbor, real *data)
{
  real whta, whtb;
  real s1,s2,s3;
  int base;
  real sDelta;
  real y,ay,by;

  if(neighbor.beforeTwo!=NONE){
    /* use quadratic interp */
    base = neighbor.beforeTwo;
    s1 = sPath[base] - s;
    s2 = sPath[base+1] - s;
    s3 = sPath[base+2] - s;
    ay = quad1dInterp(
      s1,s2,s3, data[base],data[base+1],data[base+2]);
  }else{
    /* use linear interp */
    base = neighbor.before;
    ay = data[base] + neighbor.locpos * (data[base+1]-data[base]);
  }
  if(neighbor.afterTwo!=NONE){
    /* use quadratic interp */
    base = neighbor.before;
    s1 = sPath[base] - s;
    s2 = sPath[base+1] - s;
    s3 = sPath[base+2] - s;
    by = quad1dInterp(
      s1,s2,s3, data[base],data[base+1],data[base+2]);
  }else{
    /* use linear interp */
```

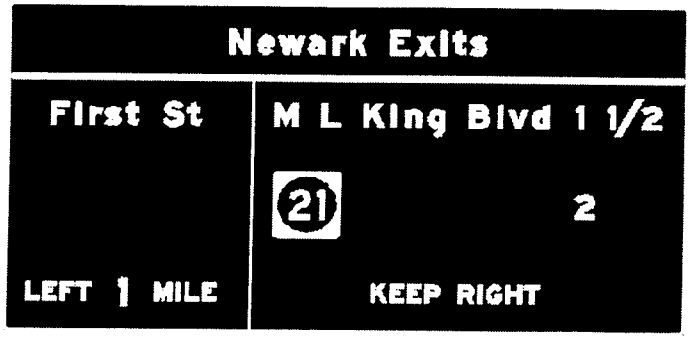
```
    base = neighbor.before;
    by = data[base] + neighbor.locpos * (data[base+1]-data[base]);
  }

  /* do weighted averaging */

  sDelta = sPath[neighbor.after] - sPath[neighbor.before];
  if(sDelta == 0.0){ /* just in case this happens ... */
    whta = 0.5; whtb = 0.5;
  }else{
    whta = 1.0 - (s-sPath[neighbor.before])/sDelta;
    whtb = 1.0 - whta;
  }
  y = whta*ay + whtb*by;

  return y;
}

/*-------------------------------------------------------------------------
quad1dInterp -- do 1d quadratic function interpolation
-------------------------------------------------------------------------*/

real
quad1dInterp(real s1, real s2, real s3, real f1, real f2, real f3)
{
  real f;
  f = s2*s3*s3*f1 - s2*s2*s3*f1 - s3*s3*s1*f2
    + s3*s1*s1*f2 + s2*s2*s1*f3 - s2*s1*s1*f3;
  f = -f / ((s3-s1)*(s2-s1)*(s2-s3));
  return f;
}

/*-------------------------------------------------------------------------
vectorForm - form vector given coordinates of endpoints
-------------------------------------------------------------------------*/

vect_t vectorForm ( coor_t i, coor_t j )
{
/*--< arguments >--*/
/* i                              /*  x,y,z-coordinates of point i (passed) */
/* j                              /*  x,y,z-coordinates of point j (passed) */

/*--< return >--*/
/* return                                 /* vector from point i to point j */

/*--< local >--*/
  vect_t vect;                             /* vector from point i to point j */

/*-------------------------------------------------------------------------*/

  vect.x   = j.x - i.x;
  vect.y   = j.y - i.y;
  vect.z   = j.z - i.z;
  vect.len = sqrt ( vect.x*vect.x + vect.y*vect.y + vect.z*vect.z );

  return ( vect );
}

/*-------------------------------------------------------------------------
vectorCross - compute vector cross product of two vectors
-------------------------------------------------------------------------*/

vect_t vectorCross ( vect_t v1, vect_t v2 )
{
/*--< arguments >--*/
/* v1                /* first of two vectors to be cross producted (passed) */
/* v2               /* second of two vectors to be cross producted (passed) */
```

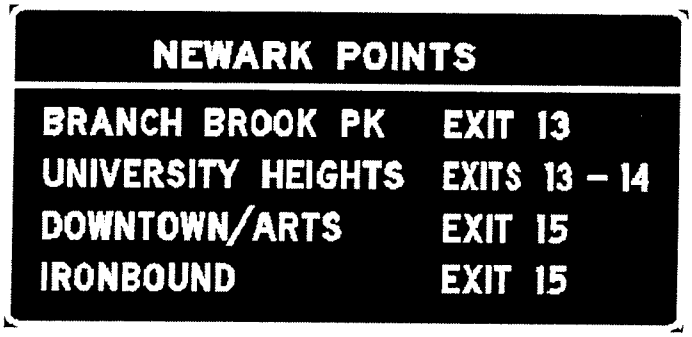
```
/*--< return >--*/
/* return                                                  /* cross product vector */

/*--< local variables>--*/
  vect_t cross;                                           /* cross product vector */

/*-------------------------------------------------------------------------------*/

  cross.x   = v1.y * v2.z - v1.z * v2.y;
  cross.y   = v1.z * v2.x - v1.x * v2.z;
  cross.z   = v1.x * v2.y - v1.y * v2.x;
  cross.len = sqrt ( cross.x*cross.x + cross.y*cross.y + cross.z*cross.z );

  return ( cross );
}

/*-------------------------------------------------------------------------------
vectorDot - compute dot product of two vectors
-------------------------------------------------------------------------------*/

real vectorDot ( vect_t v1, vect_t v2 )
{
/*--< arguments >--*/
/* v1                  /* first of two vectors to be dot producted (passed) */
/* v2                 /* second of two vectors to be dot producted (passed) */

/*--< return >--*/
/* return                                        /* dot product of two vectors */

/*--< local >--*/
  real dot;                                     /* dot product of two vectors */

/*-------------------------------------------------------------------------------*/

  dot = v1.x*v2.x + v1.y*v2.y + v1.z*v2.z;
  return ( dot );
}

/*-------------------------------------------------------------------------------
vectorUnit - convert vector to a unit vector
-------------------------------------------------------------------------------*/

vect_t vectorUnit ( vect_t vect )
{
/*--< arguments >--*/
/* vect                  /* vector to be normalized (wrt to 2-norm) (passed) */

/*--< return >--*/
/* return                                          /* the unit vector formed */

/*--< local >--*/
  char *err_msg;                                    /* error message string */
  vect_t unit;                                     /* the unit vector formed */

/*-------------------------------------------------------------------------------*/

  /* compute the length of the vector being normalized instead of */
  /* assuming that the length is in the vect_t vector structure.  */

  vect.len = sqrt ( vect.x*vect.x + vect.y*vect.y + vect.z*vect.z );

  unit.x   = vect.x / vect.len;
  unit.y   = vect.y / vect.len;
  unit.z   = vect.z / vect.len;
  unit.len = 1.0;

  return ( unit );
}
```

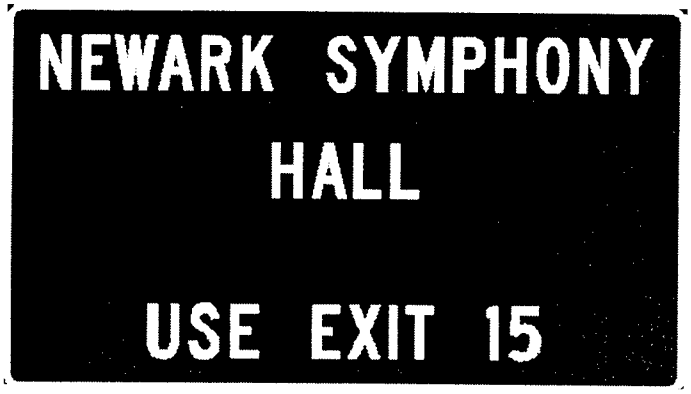
```
/*-------------------------------------------------------------------------
strClone -- clone a string
-------------------------------------------------------------------------*/
char
*strClone(char *string)
{
  size_t len;
  char *buf;
  len = strlen(string)+1;
  buf = Malloc(len);
  memcpy (buf, string, len);
  return buf;
}
```

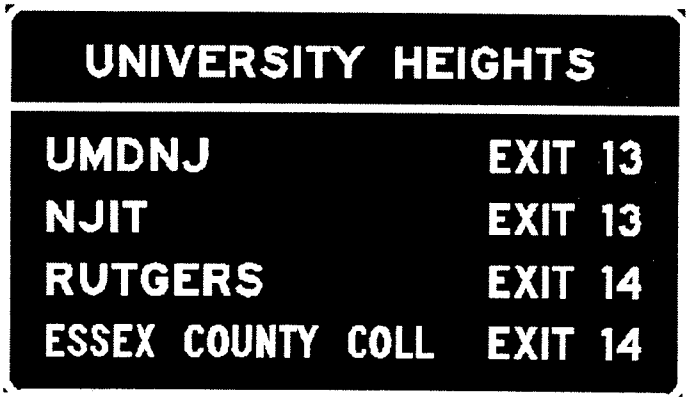
APPENDIX B – SIGN TEXTURE MAPS FOR I-280 AND RT. 21 RAMP

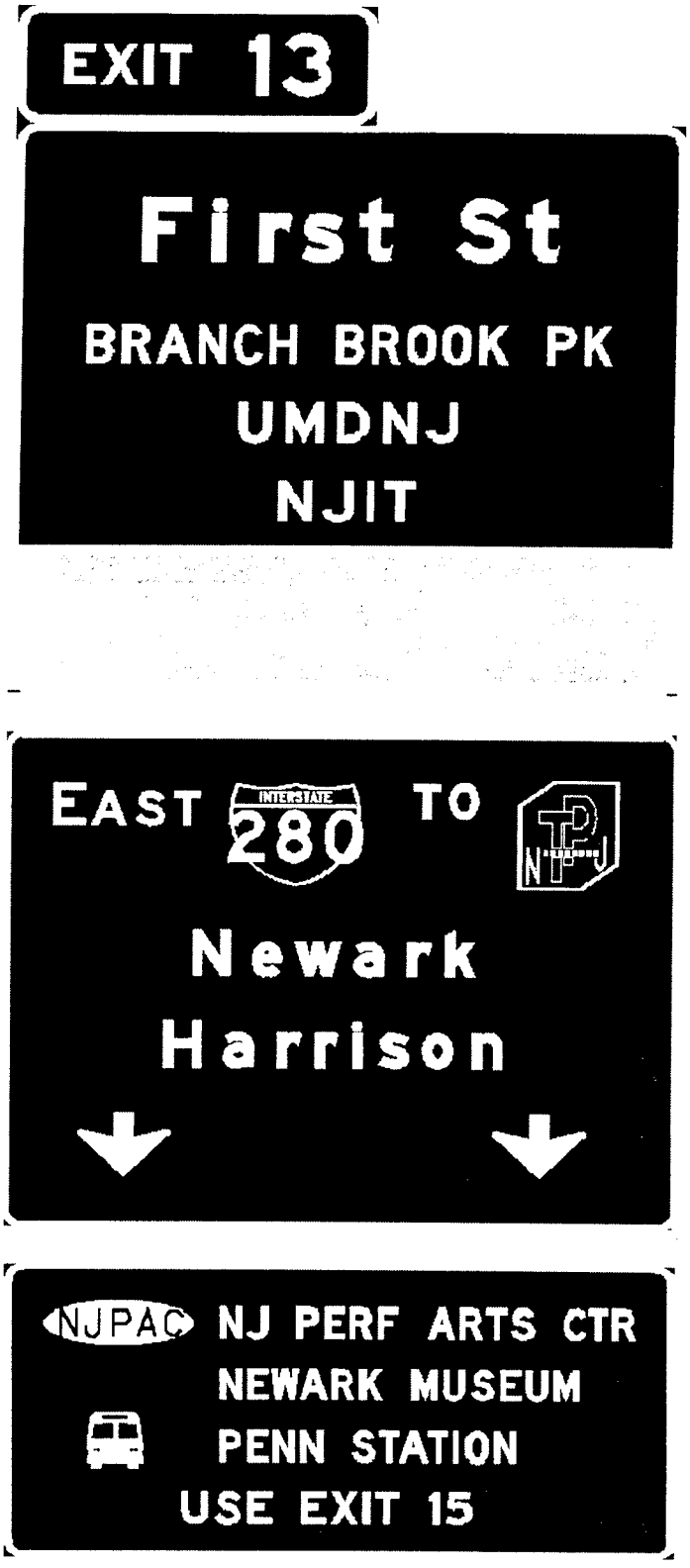
This appendix contains the texture maps that were used to render the proposed signs for the I-280 and Rt. 21 ramp drive-through visualizations. The images are presented below in approximately the same order in which they appear in the visualizations.


Newark Exits
First St
M L King Blvd 1 1/2
21
2
LEFT 1 MILE
KEEP RIGHT

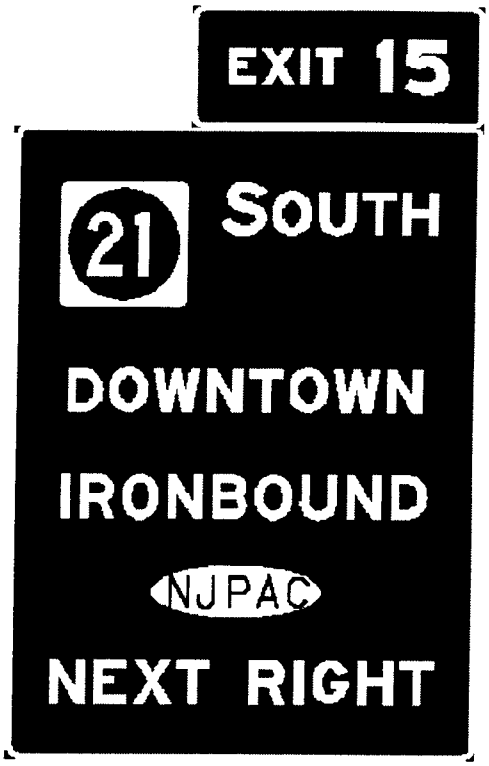

NEWARK POINTS
BRANCH BROOK PK EXIT 13
UNIVERSITY HEIGHTS EXITS 13 - 14
DOWNTOWN/ARTS EXIT 15
IRONBOUND EXIT 15

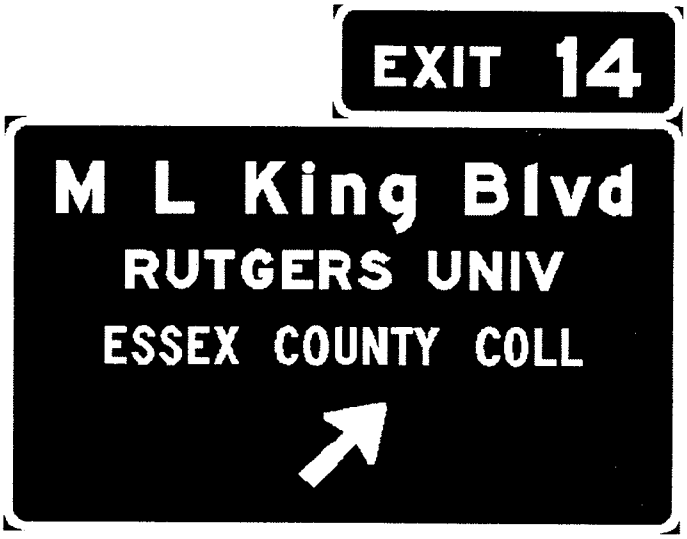

NEWARK SYMPHONY HALL
USE EXIT 15

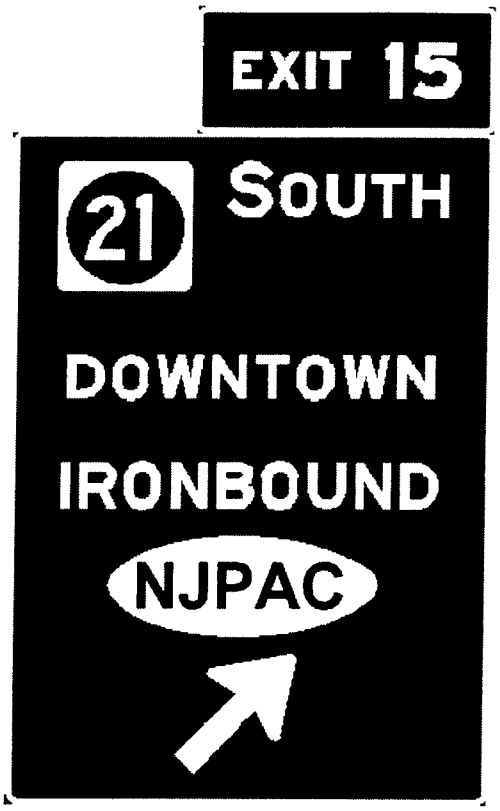

UNIVERSITY HEIGHTS
UMDNJ EXIT 13
NJIT EXIT 13
RUTGERS EXIT 14
ESSEX COUNTY COLL EXIT 14


EXIT 13
First St
BRANCH BROOK PK
UMDNJ
NJIT
EAST
INTERSTATE
280
TO
NJTP
Newark
Harrison
NJPAC
NJ PERF ARTS CTR
NEWARK MUSEUM
PENN STATION
USE EXIT 15


DRAWBRIDGE
AHEAD
3/4 MILE
EXIT 14
M L King Blvd
RUTGERS UNIV
ESSEX COUNTY COLL
NEXT RIGHT
DRAWBRIDGE
AHEAD
1/2 MILE


EXIT 15
21
SOUTH
DOWNTOWN
IRONBOUND
NJPAC
NEXT RIGHT


EXIT 14
M L King Blvd
RUTGERS UNIV
ESSEX COUNTY COLL

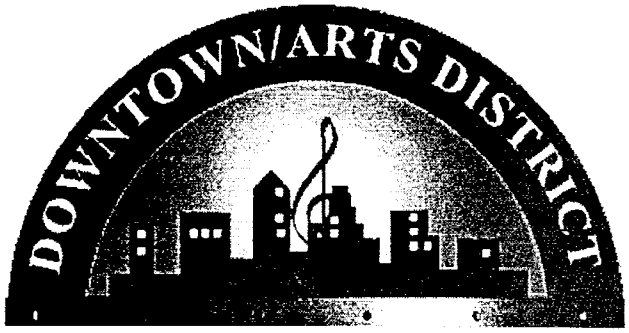


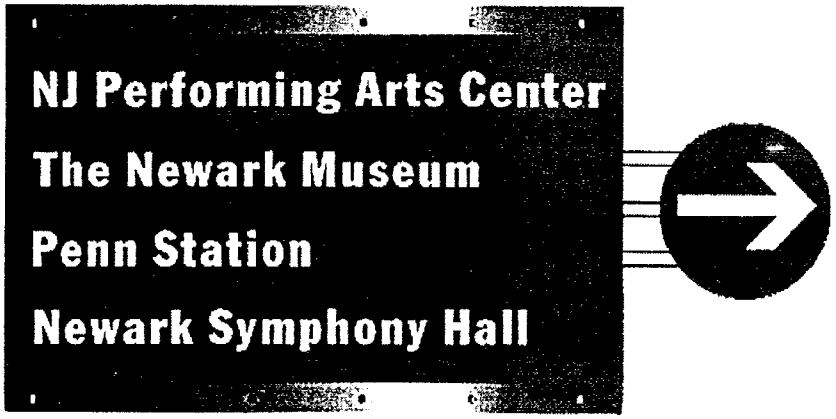
EXIT 13

EXIT 14

EXIT 15


DOWNTOWN/ARTS DISTRICT
Welcome


DOWNTOWN/ARTS DISTRICT


NJ Performing Arts Center
The Newark Museum
Penn Station
Newark Symphony Hall