Vectorized initial-value ray tracing for arbitrary velocity models

Jos van Trier

ABSTRACT

Iterative velocity-inversion methods often require tracing many rays. Consequently, vectorizing the ray tracing algorithm can significantly reduce the computational cost of these inversion methods. It is possible to vectorize a initial-value ray tracing method that is based on the numerical integration of the system of ray tracing equations. The method can handle general velocity models. A spline representation of the velocity model is used in the solution of the ray tracing equations.

INTRODUCTION

Ray tracing is one of the cheaper seismic data processing techniques. However, it can become prohibitively expensive in applications where many rays need to be traced—as in seismic tomography and other iterative migration or velocity-inversion schemes. Reducing the computational cost becomes also important if the velocity model or seismic data need to be interpreted repeatedly during an interactive velocity inversion.

Most of the cost of ray tracing methods arises not from calculating the propagation of the rays, but from checking their position with respect to the velocity model and reflector boundaries. These checking operations are not vectorizable, and, consequently, on vector computers the performance of ray tracing algorithms is often far from optimal.

Here I present a ray tracing method that is vectorizable. The method applies to the initial-value problem: it traces a fan of rays starting from a given point, and it models only transmitted, or possibly turning rays. The algorithm is vectorized over the index of the rays in the fan. The velocity model can be arbitrary, and is

most conveniently represented by spline functions. An explicit specification of the reflector boundaries is not necessary.

The algorithm is efficient when many rays have to be traced, and when the ray tracing does not have to solve two-point (boundary-value) problems, as in velocity-inversion schemes or Kirchhoff-migration programs where ray fans are shot up or down in the velocity model. However, if an exact calculation of rays going from a shot to a reflector and back to several geophones is needed, other methods are probably more useful (see Langan et al., 1985; Keller and Perozzi, 1983).

RAY TRACING METHOD

System of ray tracing equations

The ray tracing method that I have implemented is based on the ray tracing system of first-order partial-differential equations, derived from the Eikonal equation by the method of characteristics (see Nolet (1987) and Červený (1987) for more details):

$$\frac{dx_i}{ds} = \frac{p_i}{w}, \quad \frac{dp_i}{ds} = \frac{\partial}{\partial x_i}(w). \tag{1}$$

Here w is the slowness, $x_i = x_i(s)$ $(i = 1,2,3)$ are the coordinates of the ray as a function of s, the arclength along the ray, and $p_i(s)$ are the components of the slowness vector $\vec{p}$. The traveltime $T(s)$ along the ray is given by:

$$\frac{dT}{ds} = w. \tag{2}$$

The slowness vector is perpendicular to the wavefront ($\vec{p} = \nabla T$), and must satisfy:

$$|\vec{p}|^2 = \sum_{i=1}^{3} p_i^2 = w^2. \tag{3}$$

The system of ray equations together with equation (2) can be solved by a standard numerical integration method. I use a fourth-order Runge-Kutta method. The method propagates the properties of the ray (x_i, p_i, and T) over an increment in arclength by combining the information from several Euler-style steps (each involving one evaluation of the right-hand sides of equations (1) and (2)), and then uses the information obtained to match a fourth-order Taylor series expansion of the ray variables at the current position of the ray. For the purposes of this paper it suffices to treat the Runge-Kutta integration as a simple finite-difference update, e.g., for the traveltime:

$$T_{n+1} = T_n + h\,(\frac{\Delta T}{\Delta s})_n \tag{4}$$

with $T_n = T(s_n)$, the traveltime at the current end position of the ray, and $T_{n+1} = T(s_n + h)$, the traveltime after incrementing the arclength by h. Note that $(\frac{\Delta T}{\Delta s})_n$

is a linear combination of several evaluations of the right-hand side of equation (2), calculated for different arclength increments; it is not just a single term found by calculating w (equation (2)) at the current end position of the ray, as it would be for a forward finite-difference calculation.

To solve the system of equations, initial conditions must be given in the form of the starting point of the ray, its initial traveltime, and its initial direction.

Vectorization

Normally the system of ray equations is solved for one ray at a time. For two reasons this makes it hard, or even impossible, to vectorize. First, the numerical integration in the Runge-Kutta method is recursive. Second, each time the arclength is incremented, certain non-vectorizable operations must be done, namely, checking if the ray goes out of the velocity model, if it passes a given depth level, or if the traveltime exceeds a specified limit.

Instead of tracing ray by ray, one can start with a fan of rays, and propagate in one step the properties of all the rays in the fan over an increment in arclength. The numerical integration of the ray equation system can now be vectorized over the index of the rays in the fan; instead of evaluating the right-hand sides of equations (1) and (2) and updating the ray properties for one ray, I calculate the right-hand sides and update the properties for all rays in one loop over the ray index.

The vectorization does not require much extra computer memory: the different variables of the ray are now vectors—which have the number of rays in the fan as dimension—instead of scalars. For the Runge-Kutta integration some extra temporary arrays are necessary that also have as dimension the number of rays in the fan.

Non-vectorizable operations such as the ones mentioned above can be performed outside the ray-index loop. Some care is needed to avoid propagating "bad" rays (e.g., rays going out of the model, or turning rays). Normally, such rays are abandoned, and the next ray is calculated. However, now the ray variables are organized in vectors of fixed width, and it is not possible to "drop" rays as the fan is propagated. Therefore, the increment in arclength, h, is changed from a scalar to a vector. Initially, all the elements of the vector are equal to h. As soon as a ray goes out of the model, or reaches a given depth or traveltime, the increment for that ray is set to zero. When the ray variables are updated in the Runge-Kutta integration (equation (4)), the ray in question will not propagate, but will stop at its current location.

Of course, if many rays are stopped, this procedure will become inefficient as calculations are done for rays that do not move. However, in general just a small part of the ray fan will go out of the model, and the time gained by vectorizing the ray tracing is generally much larger than the time lost by unnecessary calculations.

VELOCITY MODEL

Splines

An accurate estimate of the gradient of the slowness-field (used in equation (1)) is necessary to trace rays in a velocity model with high velocity contrasts. Therefore, I have chosen to use cubic B-splines (Inoue, 1986) to parametrize the slowness model, which enables me to calculate the slowness gradient at each point in the model. Another consideration for using splines is that a wide variety of slowness models can be represented by few spline coefficients. This means that the number of parameters in an optimization scheme that inverts for slowness can be reduced considerably.

The spline representation of a two-dimensional slowness model is:

$$w(x_1, x_2) = \sum_{i=1}^{N} \sum_{j=1}^{M} c_{ij} \, F_i(x_1) \, G_j(x_2), \tag{5}$$

with N and M the number of spline knots in the x_1- and x_2-direction. F_i and G_j are the spline functions at the ith knot in the x_1-direction and the jth knot in the x_2-direction, respectively.

In practice, the slowness function is derived from a gridded velocity model. The spline coefficients are found by fitting splines to the reciprocal of the velocity values at the grid points (Inoue, 1986). Although the spline representation may deviate somewhat from the gridded model (the fit is not exact at each point), the errors in ray positioning are usually small. Such errors are negligible when the ray tracing is used in velocity inversion schemes where the velocity model is only known approximately.

To speed up the ray tracing even more, I precompute the spline functions and their derivatives and store them in a table. Consequently, an evaluation of slowness or its derivative consists only of a look-up and linear combination of precomputed spline terms. In this manner, the cost of calculating velocity values and their derivatives is comparable to that for a gridded model with linear interpolation.

Boundaries

Boundaries in the velocity model are not explicitly specified. Sharp contrasts in velocity are simply obtained by fitting splines to the velocity jumps, although this can lead to oscillations in the model. To avoid the latter, the boundary is given a small width. The velocity is linearly increased or decreased in the boundary to match the values at either side of it, and only then is it fitted by splines. An accurate specification of boundaries is often impossible, anyway, because of the limited resolution in the seismic image of the subsurface (see also Van Trier, 1988). Langan et al. (1985) discuss the effects of smoothing velocity contrasts in more detail.

Tracking reflector boundaries is part of many ray tracing methods, because reflection and transmission angles need to be calculated at the boundaries. In the ray tracing method described here, boundaries need not be specified, simplifying the computation and bookkeeping. However, the lack of boundaries in the velocity model makes it impossible to calculate reflected waves; the ray tracing is limited to transmitted, or possibly turning rays. Still, reflected rays can be computed approximately by tracing rays up from the reflector, and selecting the appropriate up and downgoing rays later, using interpolation if the rays do not end at the exact locations. The next section describes this situation in more detail.

EXTENSION TO TWO-POINT RAY TRACING

In some applications the described method can be used in an approximate two-point ray tracing method. For example, if one is interested in seismic attributes at certain points in depth, rays can be traced upwards from these points to the surface. The calculated traveltimes can be used together with the arrival positions of the rays to select from the seismic data the features of interest. This procedure is "semi-two-point": rays are not traced explicitly from one point to another, but from one point to a specific level (the surface). Stopping the rays at a given level is done as described in the section on vectorization.

Of course, rays generally will not arrive at source or receiver locations. However, because the ray tracing is vectorized over the number of rays in the fan, the density of the rays in the fan can be increased without much extra cost, and selecting the nearest ray to a source or receiver is often accurate enough for many applications. Traveltimes can be calculated by quadraticly interpolating the traveltimes between the 3 nearest rays. Alternatively, the seismic data can be interpolated instead of the ray variables: seismic attributes at the end points of the rays are found by interpolating seismic data at the receivers nearest to those end points.

This procedure is efficient if one is interested in depth points, e.g., for analyzing constant surface location gathers after migration (Van Trier, 1988), or for calculating Green's functions in Kirchhoff migration. However, if the goal of the ray tracing is to calculate rays going from a shot to several receivers, the method may become inefficient because depth points vary for each shot-receiver pair, and are not known in advance.

EXAMPLE

As an example I trace rays through the velocity model shown in Figure 1. The model contains a high-velocity salt structure. The input is a gridded model with 165 grid points in the depth direction and 300 grid points in the lateral direction. The distance between grid points is 10 m. A spline representation of the model is constructed using 26 spline knots in both the depth and lateral direction. Figure 2

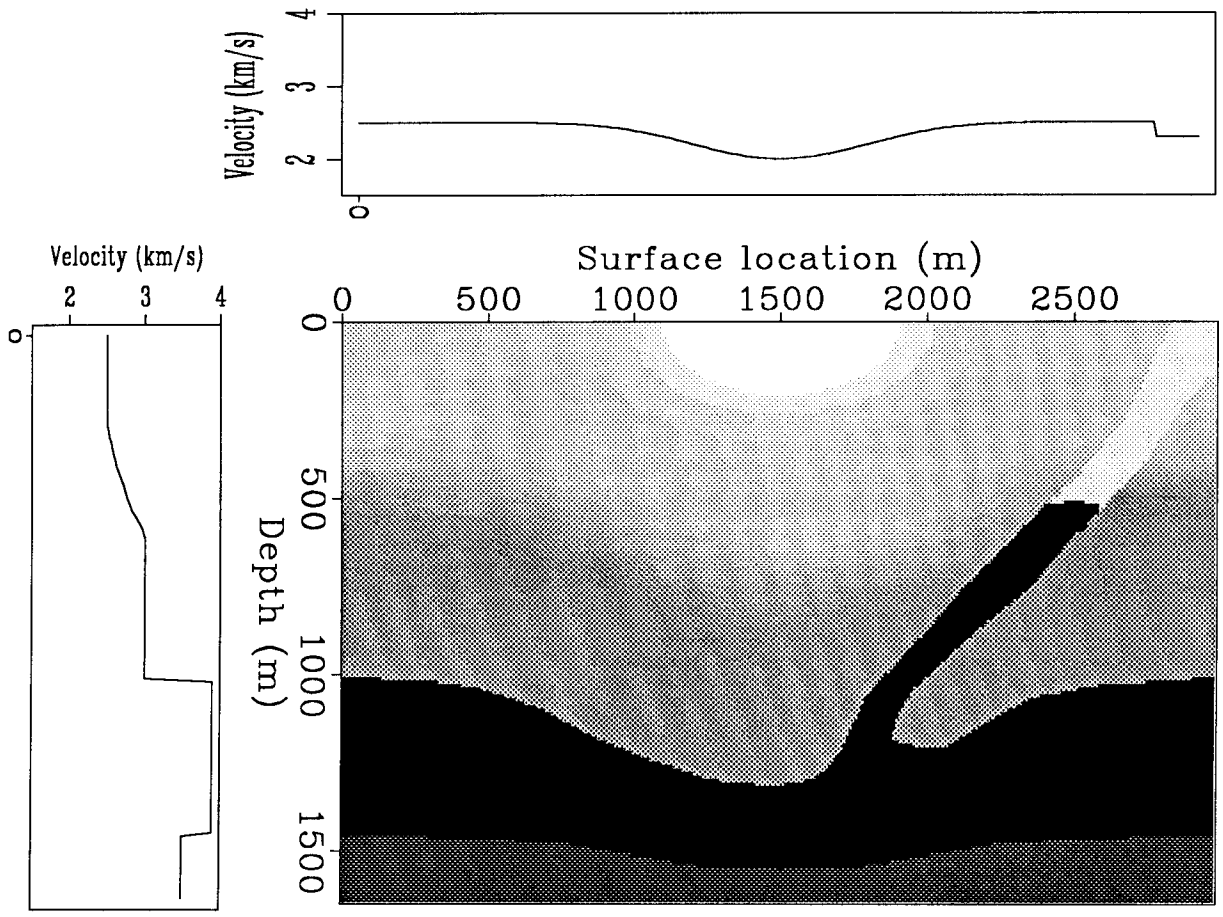


FIG. 1. Velocity model: the top structure has a velocity of 2.5 km/s, with a low-velocity region in the middle, the velocity in the second structure is smoothly varying from 2.5 km/s at the top to 3.0 km/s at the bottom, and all the other structures have constant velocities. The salt intrusion has a velocity of 3.9 km/s. Structure boundaries are shown more clearly in Figure 3. The velocity profiles in the figure are slices through the left-hand side and top of the model.

shows a depth slice through the original model and the model represented by spline functions. Before spline coefficients are calculated, sharp velocity contrasts are replaced by linear velocity gradients. The gradients extend over a vertical region with a width of 80 m. Note that some small oscillations in the velocity are still apparent near the contrasts.

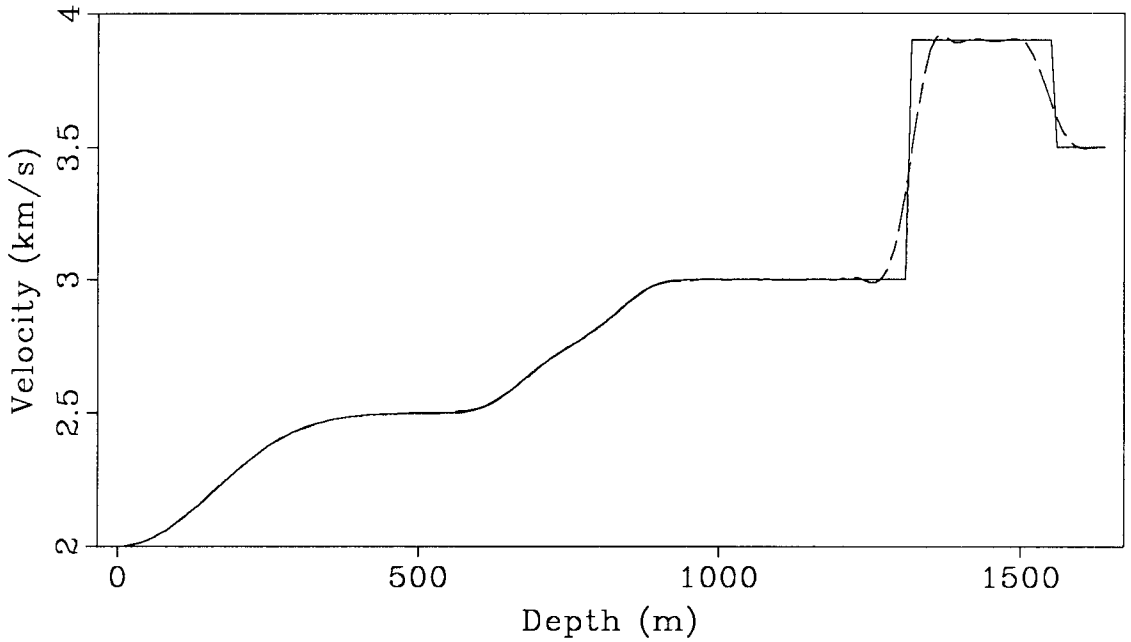


FIG. 2. Velocity profile at the surface location of 1500 m. The continuous line shows the input model, the dashed line the spline model used in ray tracing.

Figure 3a and Figure 3b show down and upgoing ray fans, respectively. The fans contain 100 rays and are calculated using 100 steps in the Runge-Kutta integration. The angles range from -30 to 30 degrees in the downgoing fan, and from -50 to 40 degrees in the upgoing fan. Note that rays start breaking before they reach the boundaries because of the mentioned smoothing of the boundaries.

The CPU-time needed to compute a ray fan with the described dimensions is about 1 s on the Convex C-1. Not vectorizing the computer code will increase the CPU-time by a factor of about 10, depending on the number of rays in the fan.

SUMMARY

The described ray tracing method is a useful tool in velocity inversion methods because it is fast, it can handle general velocity models, and it naturally fits in with a parametrization of the velocity model in spline functions. The method is

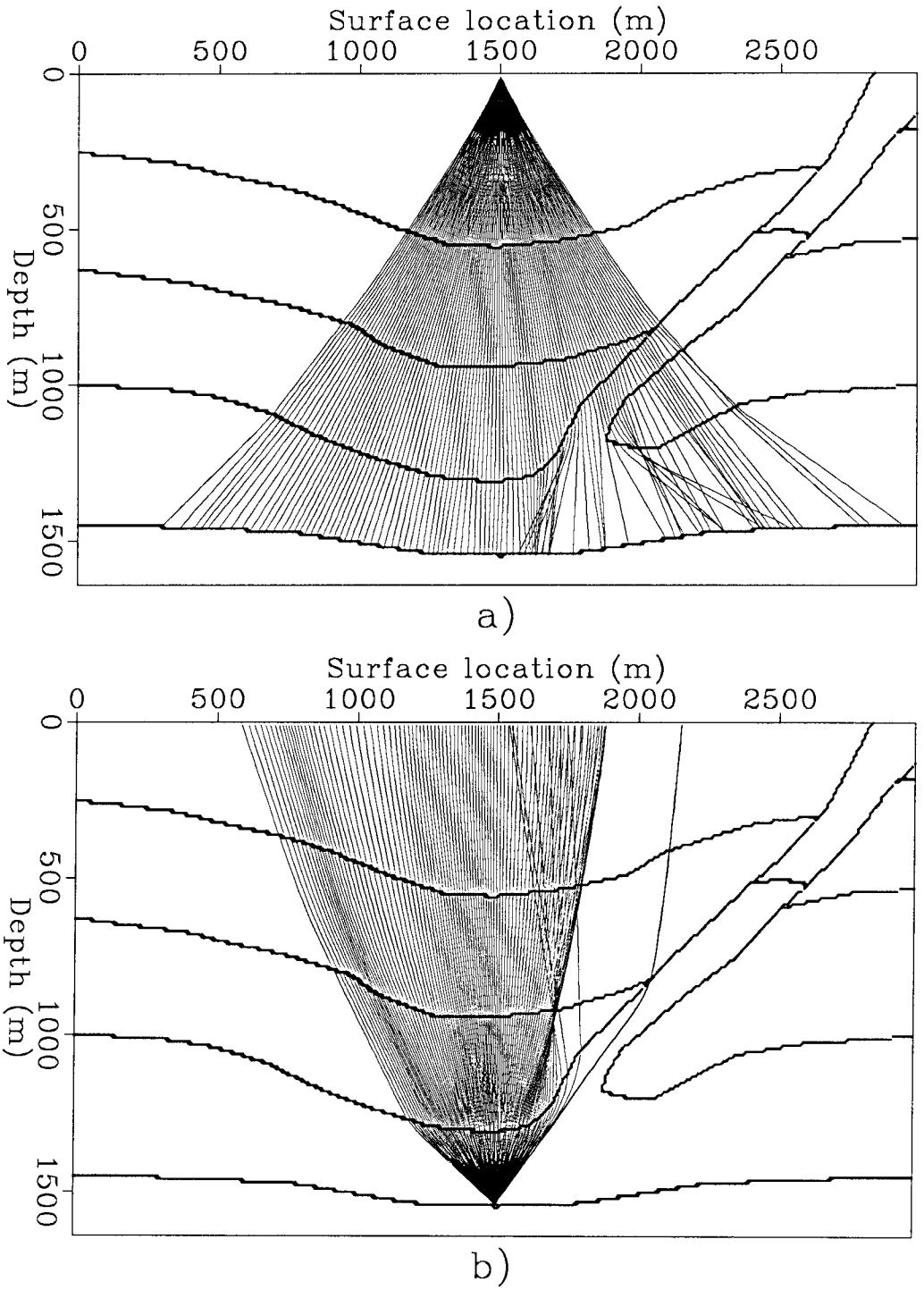


FIG. 3. Ray tracing: a) rays traced from surface to reflector; b) rays traced from depth point to surface. Reflector boundaries are not explicitly used in ray tracing; in Figure a the deepest reflector needs to be specified to determine where to end the rays.

limited to initial-value ray tracing, although it is possible to incorporate it in an approximate two-point scheme.

REFERENCES

Červený, V., 1987, Ray tracing algorithms in three-dimensional laterally varying layered structures: Seismic Tomography, edited by G. Nolet, Riedel Publishing Company, 99–134.

Inoue, H., 1986, A least-squares smooth fitting for irregularly spaced data: finite-element approach using the cubic B-spline basis: Geophysics, **51**, 2051–2066.

Keller, H.B., and Perozzi, P.J., 1983, Fast seismic ray tracing: SIAM J.Appl.Math., **43**, 981–992.

Langan, R.T., Lerche, I. and Cutler, R.T., 1985, Tracing of rays through heterogeneous media: an accurate and efficient procedure: Geophysics, **50**, 1456–1465.

Nolet, G., 1987, Seismic wave propagation and seismic tomography: Seismic Tomography, edited by G. Nolet, Riedel Publishing Company, 1–24.

Van Trier, J., 1988, High-resolution velocity inversion of migrated data after structural interpretation: SEP–**57**.

Stop 29 — **Great Basin, Mono Lake**

Mono Lake lies in an active volcanic caldera that has errupted several times in the past several million years. There have been a variety of different volcanic rocks extruded from darker basaltic flows in the north to the lighter rhyolitic flows in the south. Rapid cooling of the flow results in glassy obsidian. When there is trapped gas volatiles, then the rocks forms as porous, abrasive obsidian.

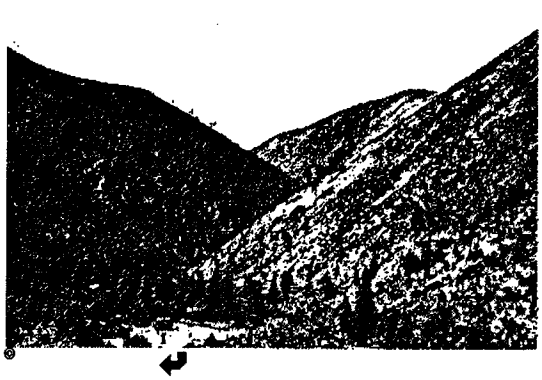
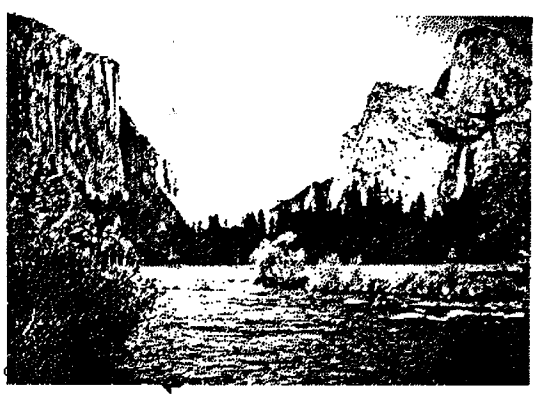
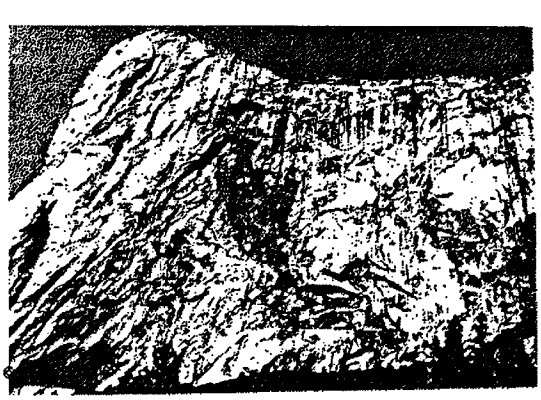
The rich calcium carbonate content of Springs feeding Mono lake has coated the volcanic rock with whitish limestone. Little mounds of limestone formed at the springs themselves. Off in the distance you can see several old shoreline levels from wetter glacial times. The water level continues to fall these

Stop 30 — **Great Basin, Bloody Canyon**

Bloody Canyon is the first Canyon south of Lee Vining Canyon that has a long train of moraine deposits extending downward into the lowlands from the mountain front. The till of an older Sherwin stage may be seen beneath those of the Tioga and Tahoe stages.

Stop 31 — **Great Basin, Mono Craters**

Mono Craters appears as several light colored circular domes. The surface of much of the craters is covered with blocky rubble of lava, much of it consisting of dark volcanic glass that displays fine flow laminae and contains frothy pumice.

The volcanism took place in a repititous sequence of events in different parts of the extensive Mono Craters complex. Three stages (in the cartoon) can be discerned. The first stage involves development of explosion pits consisting of conical depressions whose slides slope inward. The second stage involved the rise of a stiff, cylindrical, essentially solid column of obsidian that formed a vicious dome in the floor of the explosion pit. As the obsidian continued to ascend, it

cartoon